Y KNOX LIBRARY
L POSTGRADUATE SCHOOL
REY, CALIFORNIA 93943-5002

THESIS
D6326

COMPUTER SIMULATION OF SHIPBOARD ELECTRICAL DISTRIBUTION SYSTEMS

by

NORBERT HENRY DOERRY

B.S., Electrical Engineering
United States Naval Academy
(1983)

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREES OF
NAVAL ENGINEER
and
MASTER OF SCIENCE
in
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
May, 1989

© Norbert H. Doerry, 1989

Signature of Author _____
Departments of Ocean Engineering and
Electrical Engineering and Computer Science
12 May 1989

Certified by _____
James L. Kirtley, Thesis Supervisor
Associate Professor of Electrical Engineering

Certified by _____
Paul E. Sullivan, Thesis Reader
Associate Professor of Naval Architecture

Accepted by _____
A. Douglas Carmichael, Chairman
Associate Professor of Naval Architecture

Accepted by _____
Arthur C. Smith, Chairman
Electrical Engineering and Computer Science
Department Committee

T244067

Thesis
D6326
c.1

NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943

COMPUTER SIMULATION OF SHIPBOARD ELECTRICAL DISTRIBUTION SYSTEMS

by

NORBERT H. DOERRY

Submitted on May 12, 1989 in partial fulfillment of the requirements for the degrees of Naval Engineer and Master of Science in Electrical Engineering and Computer Science.

ABSTRACT

Shipboard electrical distribution systems are changing significantly with the introduction of solid state frequency converters, introduction of electric propulsion and integrated electric drive, and the possibility in the future of large combat systems pulsed loads. Existing computer tools for analyzing power systems have difficulty simulating these changing conditions. To assist in the evaluation and analysis of future shipboard electrical distribution systems, the Shipboard Electrical Plant SImulation Program (SEPSIP) was developed.

The key feature of SEPSIP is its use of implicitly defined input variables and implicit variables which allow for every element of the simulation to be mathematically isolated from every other element. When the constitutive laws of an element are satisfied by an appropriate set of input variables, all of the implicit variables have zero value. The network description generates the input variables based on the network topology and the results of a Newton-Raphson iterative scheme. The key advantage to this method is that the network description of a node closely models an actual electrical node.

To demonstrate the abilities of SEPSIP, several simulations involving synchronous generators, induction motors, and voltage regulator dynamics were conducted. In all simulations, SEPSIP provided results that matched data generated by other simulation methods.

Thesis Supervisor: James L. Kirtley
Title: Associate Professor of Electrical Engineering

The author hereby grants to the United States Government and the Massachusetts Institute of Technology permission to reproduce and to distribute copies of this thesis document in whole or in part.

_____

Norbert Henry Doerry

ACKNOWLEDGEMENTS

I am grateful to the Unites States Navy which has afforded me this opportunity to pursue a post graduate education and complete this thesis.

I am also grateful to my thesis advisor, Professor James Kirtley, for his patience, encouragement, and timely guidance

I express my thanks to the numerous individuals at the David Taylor Research Center and Naval Sea Systems Command who provided me with much insight with regards to the issues involved with shipboard electrical distribution systems.

I would also like to thank the men and women of MIT's Project ATHENA who often took time from their busy schedules to answer my questions concerning the UNIX operating system and the C programming language.

Table of Contents

CHAPTER 1 INTRODUCTION 11
1.1 Thesis Objectives 11
1.2 Modelling Shipboard Electrical Power Systems 12
1.2.1 Elements : Constitutive Equations 12
1.2.2 Networks : Nodal Equations 13
1.3 Power System Analysis Computer Tools 14
1.3.1 Desired Features 14
1.3.2 Simulation Programs for Land Based Power Utilities 14
1.3.2.1 EMTP 14
1.3.2.2 POSSIM 15
1.3.2.3 MANSTAB 15
1.3.2.4 MANTRAP 16
1.3.2.5 LOTDYS 16
1.3.3 Software Packages for Systems of Nonlinear Equations 16
1.3.3.1 CSMP 17
1.3.3.2 SIMNON 18
1.3.3.3 ACSL 18
1.3.4 SEPSIP (Shipboard Electrical Plant SImulation Program) 18
1.4 Significance of Thesis 19
1.5 Outline of Thesis 19

CHAPTER 2 THEORY OF THE COMPUTATIONAL METHOD 21
2.1 General Strategy 21
2.2 Device Definitions 23
2.2.1 Input Variables : Interacting with the Network 24
2.2.1.1 Voltage Subnodes 25
2.2.1.2 Reference Voltage Subnodes 25
2.2.1.3 Current Subnodes 25
2.2.1.4 Reference Current Subnodes 25
2.2.2 Parameters 26
2.2.3 Implicit Definition of Input Variables 26
2.2.4 Implicit Variable Selection : Rotating and Translating Axes 26
2.2.5 Data Storage : State Variables 29
2.2.6 External Inputs 29
2.2.7 External Outputs 29
2.2.8 Integration Techniques 29
2.2.9 Modelling Transfer Functions 31
2.2.10 Jacobian Construction 31
2.3 ELEMENT DESCRIPTIONS 32
2.4 NETWORK Description 33
2.4.1 Voltage Subnodes 33
2.4.2 Current Subnodes 33
2.5 Reference Subnodes 34
2.6 Conducting the Simulation 35
2.6.1 Setup 36
2.6.2 Initialization 37
2.6.3 Updating External Inputs 37
2.6.4 Balancing the System 38
2.6.4.1 Calculating Implicit Variables 38
2.6.4.2 Manufacturing System Jacobian Matrix 38
2.6.4.3 Calculating Variable Corrections via Newton-Raphson 41
2.6.5 Printing Results 41

2.6.6 Updating State Variables and Time Counter 41
2.6.7 Potential Problems 42
2.6.7.1 Numerical Instability 42
2.6.7.2 Singular Jacobian Matrix 42
2.6.7.3 Non-Unique Solutions 42

CHAPTER 3 SEPSIP SHIPBOARD ELECTRICAL PLANT SIMULATION PROGRAM 44
3.1 Introduction 44
3.2 Data Entry Conventions 44
3.2.1 Acceptable Characters 45
3.2.2 Reserved Names 45
3.2.3 Specifying Variables and Subnodes 45
3.2.4 Numerical Entries 46
3.2.5 White Characters 46
3.2.6 Continuation Lines 46
3.2.7 Case Sensitivity 46
3.2.8 Comment Lines 46
3.3 Input File Generation 47
3.3.1 Organization -- Using INCLUDE Files 48
3.3.2 ELEMENT Description 48
3.3.3 NETWORK Description 49
3.3.4 INITIALIZATION Description 51
3.3.4.1 INITIALIZE 52
3.3.4.2 EXTERNAL [INPUTS INITIALIZATION] 52
3.3.4.3 NODE VOLTAGE [INITIALIZATION] 53
3.3.5 SIMULATION Description 53
3.3.5.1 CONVERGE 53
3.3.5.2 DELTA 54
3.3.5.3 DELTA_MIN 54
3.3.5.4 DISPLAY 54
3.3.5.5 EXTERNAL INPUTS 55
3.3.5.6 MAX_ITERATION 55
3.3.5.7 PRINT_STEP 55
3.3.5.8 REFERENCE 55
3.3.5.9 TIME_STEP 56
3.3.5.10 TMIN 56
3.3.5.11 TMAX 56
3.4 Running the Simulation 57
3.4.1 Starting SEPSIP 57
3.4.2 Command Entry Conventions 57
3.4.2.1 SEPSIP Menus 57
3.4.2.2 Concatenating Commands 58
3.4.2.3 Input Filename Specification 58
3.4.2.4 Output Filename Specification 58
3.4.3 Command Summary 59
3.4.4 Main Menu 59
3.4.4.1 c Continue 60
3.4.4.2 d Display Data 60
3.4.4.2.1 d c Change Working Directory 61
3.4.4.2.2 d d Display Device Summary 61
3.4.4.2.3 d D Display Device Data 62
3.4.4.2.4 d e Display Element Summary 62
3.4.4.2.5 d E Display Element Data 63
3.4.4.2.6 d n Display Network Summary 63

3.4.4.2.7 d q Quit 63
3.4.4.2.8 d w Write Device Data File 64
3.4.4.3 e Edit Simulation Parameters 64
3.4.4.3.1 e d Edit Display Variables 65
3.4.4.3.1.1 a Add Display Variable 65
3.4.4.3.1.2 d Delete Display Variable 66
3.4.4.3.1.3 q Quit 66
3.4.4.3.2 e j Edit Jacobian Parameters 67
3.4.4.3.3 e q Quit 67
3.4.4.3.4 e r Edit Reference Voltage Subnode 68
3.4.4.3.5 e t Edit Time Parameters 68
3.4.4.4 f File Options 69
3.4.4.4.1 f d Dump Simulation State 69
3.4.4.4.2 f i Save INITIALIZATION Section 70
3.4.4.4.3 f I Load INITIALIZATION Section 70
3.4.4.4.4 f s Save SIMULATION Section 71
3.4.4.4.5 f S Load SIMULATION Section 71
3.4.4.5 q Quit 71
3.4.4.6 s Conduct Simulation 72
3.4.4.7 u Utilities 72
3.4.4.7.1 u e Editor -> emacs 73
3.4.4.7.2 u p Plotting -> Norplot 73
3.4.4.7.3 u ? List Directory 73
3.4.4.7.4 u % Execute System Command 74
3.4.4.7.5 u + Screendump to default printer 74
3.5 Special Considerations 75
3.5.1 Designing the Network 75
3.5.2 Selection of Time Increments 75
3.5.3 Using the SIMULATION File 77
3.5.4 Using the INITIAL File 77
3.6 Adding DEVICE Descriptions 77

CHAPTER 4 DEVELOPMENT OF SHIPBOARD ELECTRICAL COMPONENT MODELS 78
4.1 Transmission Line Model 79
4.2 Resistive - Reactive Load Model 82
4.3 Synchronous Machine Model 85
4.4 Speed Governor 91
4.5 Voltage Regulator Model 93
4.6 Induction Motor Model 97
4.7 Three Phase Switch Model 102
4.8 Circuit Breaker Model 105

CHAPTER 5 SIMULATION RESULTS 109
5.1 50 HP Induction Motor Start Up 110
5.2 500 HP Induction Motor Start Up 114
5.3 Synchronous Generator: Switched Load 118
5.4 Synchronous Generator: Two Phase Fault 124
5.5 Synchronous Generator: Three Phase Fault 130
5.6 Paralleled Synchronous Generators: Switched Load 133
5.7 Paralleled Synchronous Generators: Switched Load 140

CHAPTER 6 CONCLUSIONS 144
6.1 Assessment of SEPSIP 144

6.2 Future Improvements 145
6.2.1 Variable Time Step 145
6.2.2 Replace Gaussian Elimination 145
6.2.3 Reuse of Jacobian Matrix 146
6.2.4 Output Variables and Output Subnodes 146
6.2.5 Action Files 146
6.2.6 Integrated Graphics 148
6.2.7 Implement external variable 'types' 148
6.2.8 Optimization for speed 149
6.2.9 Check for Recursive INCLUDE Files 149
6.2.10 Break Key 149

References 151

APPENDIX A GLOSSARY 153

APPENDIX B INSTRUCTIONS FOR ADDING DEVICES 157
B.1 Write Device Driver Routines 157
B.1.1 Arguments 157
B.1.2 Select number and types of variables 159
B.1.3 Calculate Implicit variables 159
B.1.4 Calculate State Variables 159
B.1.5 Calculate External Output Variables (optional) 160
B.1.6 Calculate Jacobian Matrix (optional) 160
B.2 Modify Device Input File (three_phase.input) 160
B.3 Modify penner.h 161
B.4 Modify Makefile 162
B.5 Recompile SEPSIP 163

APPENDIX C DEVICE DRIVER CODE 164
C.1 f_t_line_3p.c 165
C.2 f_rl_wye.c 170
C.3 f_synch_mach.c 175
C.4 f_speed_reg.c 183
C.5 f_volt_reg.c 187
C.6 f_ind_motor.c 191
C.7 f_switch_3p.c 196
C.8 f_breaker_3p.c 201
C.9 f_spst_switch.c 208
C.10 f_gen_synch_3p.c 211
C.11 penner.h 216
C.12 three_phase.input 220
C.13 one_phase.input 230

APPENDIX D MENU DRIVER CODE 235
D.1 menu.c 235
D.2 sepsip_util.menu 241

APPENDIX E PORTABILITY CONSIDERATIONS 242

APPENDIX F SEPSIP SOURCE CODE 243

Table of Figures

2.1-1 Resistor 22
2.2-1 SEPSIP Variables 24
2.2.4-1 Diode 26
2.6.4.2-1 Manufacturing System Jacobian 40
3.3-1 Sample SEPSIP Input File 47
3.3.1-1 Using the INCLUDE Keyword 48
3.3.2-1 ELEMENT Description 48
3.3.3-1 NETWORK Description 49
3.3.3-2 NETWORK Description Example 51
3.3.4.2-1 EXTERNAL INPUTS INITIALIZATION Subsection 52
3.3.4.3-1 NODE VOLTAGE INITIALIZATION Subsection 53
4.1-1 Transmission Line 79
4.2-1 Resistive - Reactive Load Model 82
4.3-1 Synchronous Machine Model 85
4.4-1 Speed Governor 91
4.5-1 Voltage Regulator 93
4.6-1 Induction Motor Model 97
4.7-1 Three Phase Switch 102
4.8-1 Circuit Breaker 105
4.8-2 Circuit Breaker States 105
4.8-3 Breaker Transform Table 1 107
4.8-4 Breaker Transform Table 2 107
4.8-5 Breaker Transform Table 3 108
5.1-2 t50.elm : Element Description File 110
5.1-1 50 HP Induction Motor 111
5.1-3 t50.net : Network Description File 111
5.1-4 t50.init : Initialization File 112
5.1-5 t50.sim : Simulation File 112
5.1-5 t50.all : Input File 112
5.1-7 RPM vs Time 113
5.1-8 Te vs RPM 113
5.1-9 Stator Current vs Time 113
5.1-10 Rotor Current vs Time 113
5.2-1 500 HP Induction Motor 114
5.2-2 t500.elm : Element Description File 114
5.2-3 t500.net : Network Description File 115
5.2-4 t500.init : Initialization File 115
5.2-5 t500.sim : Simulation File 116
5.2-6 RPM vs Time 117
5.2-7 Te vs RPM 117
5.2-8 Stator Current vs Time 117
5.2-9 Rotor Current vs Time 117
5.3-1 Synchronous Generator: Switched Load 118
5.3-2 v.elm : Element Description File 119
5.3-3 v.net : Network Description File 119
5.3-4 v.init : Initialization File 120
5.3-5 v.sim : Simulation File 120
5.3-6 RPM vs Time 122
5.3-7 Terminal Voltage vs Time 122
5.3-8 Field Voltage vs Time 122
5.3-9 Tepu vs Time 122
5.3-10 id and iq vs Time 123

5.3-11 vd and vq vs Time 123
5.4-1 Synchronous Generator: Two Phase Fault 124
5.4-2 w.elm: Element Description File 124
5.4-3 w.net: Network Description File 125
5.4-4 w.init: Initialization File 126
5.4-5 w2.sim: Simulation File 127
5.4-6 RPM vs Time 128
5.4-7 Torque PU vs Time 128
5.4-8 vd and vq vs Time 128
5.4-9 id and iq vs time 128
5.4-10 Line Voltage 129
5.4-11 Field Voltage 129
5.5-1 Synchronous Generator: Three Phase Fault 130
5.5-2 w3.sim: Simulation File 130
5.5-3 RPM vs Time 131
5.5-4 Torque PU vs Time 131
5.5-5 vd and vq vs Time 132
5.5-6 id and iq vs time 132
5.5-7 Line Voltage 132
5.5-8 Field Voltage 132
5.6-1 Paralleled Synchronous Generators 133
5.6-2 x.elm: Element Description File 133
5.6-3 x.net : Network Description File 135
5.6-4 x.init : Initialization File 136
5.6-5 x2.sim : Simulation File 137
5.6-6 RPM vs Time 138
5.6-7 Terminal Voltage vs Time 138
5.6-8 Field Voltage vs Time 138
5.6-9 Tepu vs Time 138
5.6-10 id and iq vs Time 139
5.6-11 vd and vq vs Time 139
5.7-1 Paralleled Synchronous Generators 140
5.7-2 x3.sim : Simulation File 140
5.7-3 RPM vs Time 141
5.7-4 Terminal Voltage vs Time 141
5.7-5 Field Voltage vs Time 142
5.7-6 Tepu vs Time 142
5.7-7 id and iq vs Time 142
5.7-8 vd and vq vs Time 142
B.1-1 Device Driver Routine 157
B.1-2 ELEMENT and CONNECT Structures 158
B.2-1 Device Input File Entry 160
B.4-1 UNIX Makefile Example 163

CHAPTER 1

INTRODUCTION

1.1 Thesis Objectives

Shipboard electrical power systems in the U.S. Navy are experiencing a number of significant changes. These changes include :

- Solid state frequency converters replacing motor generator sets
- The use of switched DC power supplies by many loads
- Centralized and automated power system control
- More frequent use of electronic motor controllers
- Sensitive electronic equipment requiring high quality 60 Hz. power.
- Electric propulsion
- Large combat systems pulse loads.

Considering the tremendous expense involved with constructing a modern warship, it is necessary to ensure that the incorporation of these changes into the design of the shipboard electrical generation and distribution system can be successfully accomplished with no degradation in the combat capability of the ship. Unfortunately, these changes, along with the small size of the shipboard generation system make the use of many classical methods of analyzing power systems inappropriate. A good analysis requires the recognition of the following properties of the shipboard system:

- The small number of generators (typically only one or two) with the associated small amount of rotational inertia invalidates any assumption of an 'infinite bus' operating at a constant frequency.

- The dynamics of the generator voltage regulators and speed governors have time constants of an order that are important in the study of most disturbances.

- The dynamics of paralleled generators are coupled through the communication of load

sharing and bus voltage information.

- Solid state frequency converters and switched power supplies have non-sinusoidal current characteristics.

- Solid state controllers often greatly modify or even substitute their own dynamics for the dynamics of the motor they are controlling. The controllers may also have non-sinusoidal current characteristics.

- High power pulsed loads for advanced combat systems may become a reality in the near future and deserve study.

- Integrated Electric Drive where propulsion motors and ship's service power are taken from the same distribution system can result in large transients from speed changes in the propulsion system propagating to all of the other loads on the ship.

The purpose of this thesis is to present the theory and design methodology used in the development of a computer simulation tool (**SEPSIP** : **S**hipboard **E**lectrical **P**lant **SI**mulation **P**rogram) for analyzing both the steady-state and transient behavior of shipboard electrical power distribution systems.

1.2 Modelling Shipboard Electrical Power Systems

Electrical Power systems are not unlike any other electrical network in that they are composed of electrical elements and the topological network connecting the elements. Each of the elements is defined by a number of constitutive equations that relate the voltages and currents of its own terminals. The network on the other hand, is defined by relating the voltages and terminals of different elements through Kirchhoff's voltage and current laws.

1.2.1 Elements : Constitutive Equations

The constitutive equations describing an element can be very complicated. In shipboard electrical power systems, the equations can take on the form of nonlinear differential equations or even discontinuous functions. Additionally, elements such as generators and motors require the mechanical subsystem be described in detail. Other elements, such as

switches and circuit breakers, are defined by equations depending on the current state of the element. All of these factors contribute to make shipboard electrical power systems difficult to simulate numerically.

1.2.2 Networks : Nodal Equations

The nodal equations are the mathematical expression of Kirchhoff's voltage and current laws that define the network topology. Kirchhoff's current law states that the sum of the currents entering a node must equal the sum of the currents leaving a node. Kirchhoff's voltage law states that the voltage at any node is identical for every element attached to it.[1] In themselves, the mathematical representation of Kirchhoff's laws are very simple. However, the resulting system of nonlinear equations is often **stiff** which implies that the eigenvalues of a linearization of the set of nonlinear equations fall in a range spanning several orders of magnitude.

Stiff systems can be solved numerically, but they require special care. The choice of time increments, integration methods, and simulation time are all affected by how stiff a system is. If a particular differential equation is known to have a very fast time constant, one can ignore the dynamics and always use the *final* value for the variable. If used properly, the Euler Backward method for integrating differential equations approaches the same solution. Fast modes can also be eliminated by a host of other model reduction techniques. [9] [10] [19] [28] [31] In any case, a tool designed to analyze shipboard systems must incorporate a method for dealing with stiff systems.

---

1 An expression of Kirchhoff's Voltage Law that may be more familiar is: The sum of the voltage drops across the elements of a closed loop is equal to zero. The two definitions are not exactly identical but are consistent with one another when one accepts the concept of a voltage being a potential value.

1.3 Power System Analysis Computer Tools

1.3.1 Desired Features

In light of the characteristics and potential problems associated with modelling shipboard electrical systems, a computer analysis tool should have the following capabilities:

* Ability to solve systems of nonlinear differential equations.
* Ability to handle changing network topologies due to the actions of switches and circuit breakers.
* Ability to handle discontinuous functions.
* Ability to organize the input data into a form that is recognizable as an electrical network.
* Ability to easily add or subtract elements from the network description.
* Ability to model mechanical subsystems.
* Ability to solve stiff systems.

The requirements on a simulation program that these features impose are not trivial. In fact, the author is unaware of any commercially available software package that incorporate all of the listed capabilities. The software that is available can be split into two categories: Programs used for analyzing commercial power utilities, and software packages for solving systems of nonlinear equations.

1.3.2 Simulation Programs for Land Based Power Utilities

There are a number of computer programs which can solve different aspects of the power system analysis problem. However none of these programs are optimized for analyzing shipboard systems. Here is a brief summary of several existing programs:

1.3.2.1 EMTP

The **Electromagnetic Transients Program (EMTP)** [22] is a large-scale network simulation program originally developed by the Bonneville Power Association in the 1960's. It is capable of modeling traveling waves on transmission lines, lumped linear

elements, the saturation of transformers and reactors, the dynamics of synchronous machines as well as other elements of a power network. EMTP handles stiff systems by using the Euler Backward method for integration. In general, the program is optimized for studying the interaction between the dynamics of a number of generators and the dynamics of the interconnecting transmission lines. The dynamics of the loads are not considered important. Unfortunately, the dynamics of loads are important in shipboard systems. Furthermore, EMTP was written in FORTRAN for batch processing and is not very easy to use interactively.

1.3.2.2 POSSIM

The **POwer System SIMulator (POSSIM)** [19] is a fifty machine transient stability program developed by the General Electric Company. POSSIM uses the results of a network load flow program (LOFYR: LOad Flow and Y-matrix Reduction) as a starting point for a multi-machine simulation. The program allows for dynamics only in the generators and their associated governors and prime movers. While generators and prime movers can be modeled in detail, the transmission line and load equations are purely algebraic. POSSIM also assumes frequency deviations are small. Since load dynamics are important in shipboard electrical systems and frequency deviations can become large, POSSIM's applicability is limited.

1.3.2.3 MANSTAB

The **MAchine and Network STABility (MANSTAB)** [19] program is an extension of POSSIM which also includes transmission line dynamics. It still does not allow for dynamics in any of its loads. Generally, MANSTAB is suitable for studying high speed dynamics and does not include governor or prime mover models. For this reason, MANSTAB is not suitable for simulating shipboard systems.

1.3.2.4 MANTRAP

The **MAchine and Network TRAnsient Program (MANTRAP)** [4] is a General Electric Company modification of the Bonneville Power Administration's **Network Transients Program**. It is designed to solve problems concerning the interaction between a synchronous generator, its excitation system, torsional system, and the power transmission system. MANTRAP has a major drawback in shipboard studies in that its assumption of an infinite bus does not hold.

1.3.2.5 LOTDYS

The **LOng Term DYnamic Simulator (LOTDYS)** [21] is designed to study long term transients of power systems lasting up to 5, 10, or 20 minutes. LOTDYS assumes all the generators operate at the same speed and the generator transient time constants and reactances can be ignored. LOTDYS also ignores excitation system dynamics and load dynamics. Prime mover dynamics, load shedding, and power plant auxiliaries are all modeled in detail. The constraints on LOTDYS severely limits its usefulness in studying shipboard systems.

1.3.3 Software Packages for Systems of Nonlinear Equations

Since the models of most electric machines are described as systems of linear or nonlinear equations, it seems reasonable that a general simulation program could be used to simulate the shipboard system. On closer examination however, the presently available software packages are limited in their ability to organize and interconnect several different machine models into a large network. Writing a shipboard electrical system as one totally integrated model invites the introduction of numerous programming errors as the input definition file becomes so large as to be unmanageable. Furthermore, the task of trying to add or subtract elements from the network becomes formidable. The ability to define elements in separate blocks is very important in understanding what a simulation is doing, in debugging an input file, and in making error free changes to the configuration.

Many of the nonlinear equation solving packages are unable to solve implicit equations which is a handicap when trying to interconnect different models. In electrical system simulations, the variables representing the voltages and currents at the terminals of an element must be shared with the other elements that connect to the terminals. If implicit equations are not allowed, one of two methods is normally used to effect this sharing. The first method calls for one device explicitly defining the variable while all the remaining variables implicitly define it. This method is very difficult to implement because it forces one to define variables to be either inputs (implicitly defined) or as outputs (explicitly defined). Problems arise when one tries to connect two outputs or two inputs together (i.e. connecting two generators in parallel). Since in real electrical systems there is no such thing as an input or an output (voltages and currents depend on the properties of all the elements attached to a node), this method imposes an artificial constraint on the network definition.

The other method for interconnecting element models is to define the voltages of every device to be inputs and the currents to be outputs. The voltage at a node is defined as a separate variable whose derivative is equated to function of the sum of the currents entering the node. This function should result in the node voltage having a very fast time constant. Adding the fast time constant however, makes the system stiff and difficult to solve numerically. It also adds dynamics that are purely fictional and in general, defeats the purpose of model reduction.

1.3.3.1 CSMP

IBM's **Continuous System Modeling Program (CSMP III)** [24] is a general purpose program for solving algebraic and differential equations. The program is not capable of solving implicit equations and is limited in the size of the systems it can model. For these reasons CSMP should not be used to simulate shipboard systems.

1.3.3.2 SIMNON

SIMNON is a program for **SIMulating NONlinear** systems of equations which was developed by the Lund Institute in Sweden. It is similar in many respects to CSMP and likewise suffers from its inability to solve implicit equations.

1.3.3.3 ACSL

The **Advanced Continuous Simulation Language (ACSL)** [1] is another general purpose simulation program like CSMP and SIMNON. While ACSL does have the ability to include implicit equations, its execution time slows tremendously when they are included. ACSL also requires a single input file which can become very large and unmanageable for even moderately sized power systems.

1.3.4 SEPSIP (Shipboard Electrical Plant SImulation Program)

Since none of the commercially available software was suitable for studying shipboard electrical distribution systems, the author undertook the task of developing the **Shipboard Electrical Plant SImulation Program (SEPSIP)** which incorporates all of the desired features listed in section 1.3.1. SEPSIP solves the problem of interconnecting device models by forcing all of the device electrical variables to be input variables. This approach mathematically isolates all of the elements of the network from one another. A separate network description specifies how the different input variables relate to one another. The network description provides values for all of the input variables for every element. The elements in turn, provide feedback in the form of **implicit variables** to the network description as to how well these input variables solve the constitutive equations defining the element. The manner in which this is accomplished is discussed in chapter 2.

The equations defining an electrical device are subroutines of SEPSIP written in the C programming Language. This allows for very detailed and complex models to be incorporated in simulations. It also requires a detailed knowledge of programming in C. Once a device description has been written however, its inclusion into network descriptions

is easy. The concept of using SEPSIP is that initially a number of device descriptions are written to describe the various elements of a shipboard system. Once this library of devices has been created, simulations can be conducted by constructing networks interconnecting the devices models selected from the established library.

1.4 Significance of Thesis

The discussions presented in the previous sections demonstrate the need for a computer analysis tool for simulating shipboard electrical distribution systems. SEPSIP, the program written as a part of this thesis, is capable of conducting the desired simulations if time is not a constraint. The organization and user interface of SEPSIP has been optimized to simulate electrical distribution systems such as those found onboard warships. SEPSIP still requires optimization to improve the speed in which it completes simulations. In this regard, potential improvements to SEPSIP are included in chapter 6.

The general nature in which SEPSIP organizes and solves systems of nonlinear equations has applications outside of electrical power engineering. Any physical system composed of a topological network interconnecting nonlinear dynamic elements can be modelled with SEPSIP. The author in fact, has successfully used SEPSIP to conduct a nonlinear dynamic simulation of the motions of a submarine in response to control surface deflections. For this simulation devices were created which related the motions of the bare hull and various appendages to the forces and torques on the center of the submarine. The results of this simulation correctly predicted dynamic responses that can not be derived from conventional linear theory.

1.5 Outline of Thesis

The following chapters are organized to correspond to the four design elements used to create SEPSIP. The second chapter describes the theory and strategy that define the requirements and properties of SEPSIP. The third chapter is a "user's manual" that describes how the requirements of chapter 2 were implemented. Chapter four presents eight device

descriptions created for SEPSIP to demonstrate its usefulness. Chapter five uses the devices of chapter four to conduct actual simulations and where possible, to verify the results from SEPSIP with known responses.

Chapter six provides an assessment of SEPSIP and lists a number of possible improvements for the program. The appendices provide listings of source code and instructions for adding new device descriptions to SEPSIP.

CHAPTER 2

THEORY OF THE COMPUTATIONAL METHOD

2.1 General Strategy

The principle underlying the organization of SEPSIP is that the constitutive relations of the elements of a power system should be separated from the nodal equations describing the network interconnecting the elements. In electrical systems, a **node** is the electrical connection between two or more elements. The current entering or leaving a node must conform to Kirchhoff's Current Law which states that the sum of the currents entering a node is equal to the sum of the currents leaving the node. Additionally, the voltage at a node is the same for every element attached to it. The role of the constitutive relations is to relate the voltages of the nodes an element is connected to, to the currents resulting from the element that enter and leave those same nodes. These principles are fulfilled in SEPSIP by implicitly defining all of the network voltages, currents, and other variables within the constitutive equations defining the individual elements. During each interval of the time domain simulation, the network variables are systematically varied so that Kirchhoff's current law is always satisfied and until all the implicitly defined constitutive equations are satisfied.

The advantage to this method is that each element can be treated separately from all the other elements. The element models need not be concerned with the other elements they are connected to. It is the responsibility of the network equations to provide input variables that satisfy the element's constitutive relations, and still satisfy the nodal equations. The purpose of defining the constitutive relations implicitly is to provide feedback to the network equations that indicate how far off the input variables provided by the network are from satisfying the constitutive relations. This feedback is used to make corrections to the input variables until all of the constitutive relations are satisfied.

To implicitly define the constitutive equations for the element, they are put into the form:

$$\bar{F}(\bar{x}) = 0 \qquad [1]$$

where

$\bar{x}$ = Network Variable Vector

$\bar{F}()$ = A vector operator (potentially nonlinear) that describes the constitutive equations.

If an $\bar{x}$ is chosen so that the constitutive equations are not fulfilled, then [1] will not be true. Instead, an implicit vector can be defined:

$$\bar{I} = \bar{F}(\bar{x}) \qquad [2]$$

Each element then, will have its own $\bar{x}$ and corresponding implicit vector. The role of the network is to choose appropriate $\bar{x}$ vectors that satisfy the network nodal equations and result in the $\bar{I}$ vectors having zero length.

EXAMPLE:

A resistor is a simple example for illustrating this principle of defining constitutive equations implicitly. A resistor is a device that connects two nodes and satisfies Ohm's Law. If we define the voltages at the two nodes to be v_0 and v_1 and the current entering the resistor from the two nodes to be i_0 and i_1, then the constitutive relations for the resistor are:

$$v_0 - v_1 = i_0 R$$

$$i_0 + i_1 = 0$$

where

R = Resistance

Figure 2.1-1 Resistor

Since a resistor is a linear device, the constitutive relation can be expressed as a matrix equation of the form:

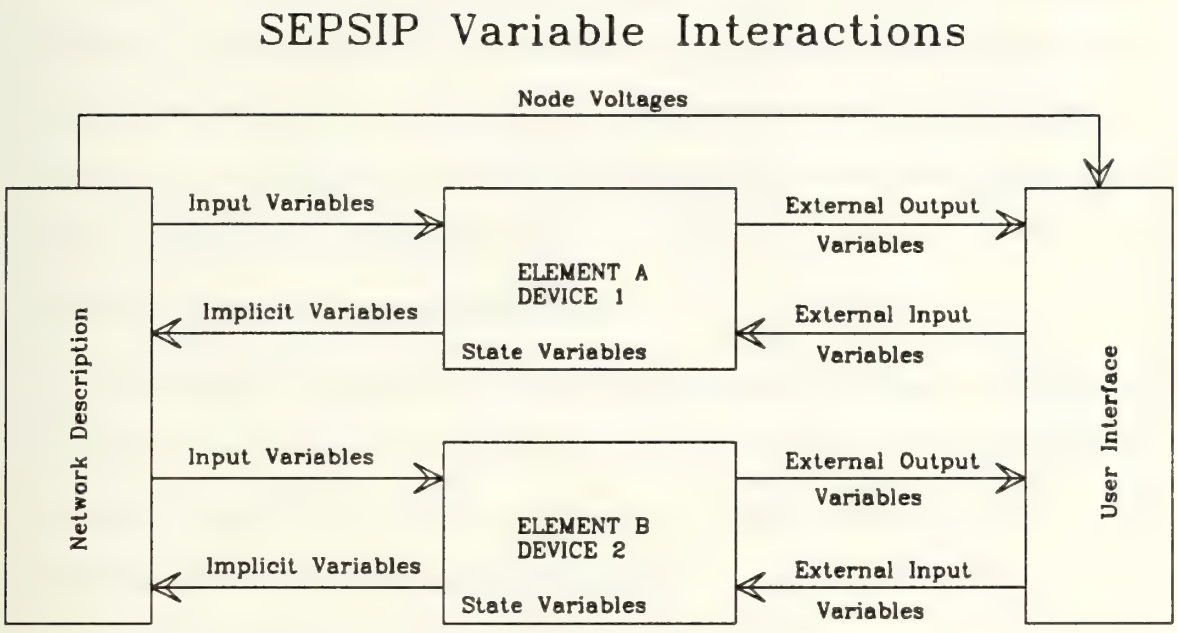
$$\bar{I} = \begin{pmatrix} 1 & -1 & -R & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ i_0 \\ i_1 \end{pmatrix}$$

Since a resistor is a linear element, the matrix in the above equation is also the Jacobian matrix for the device.

2.2 Device Definitions

In the language of SEPSIP, a **device** is a mathematical model of a piece of electrical equipment. Examples of devices are models of synchronous generators, transmission lines, breakers, induction motors, and resistive loads. A device is differentiated from an **element** in that an element is a particular example of a device. For example, element GTG1 may represent the 2000 KW gas turbine generator located in the forward engine room of a destroyer. Part of GTG1's description would be that it is a device of type KY103 which indicates which equations should be used to model GTG1. There could also be a GTG2 of type KY103. A device is characterized by the equations which relate the variables that represent the interaction of the device with everything else external to itself. Examples of device variables include voltages, currents, speeds, forces, torques, position of switches, temperature, and pressure. These variables can be organized into a number of categories according to the nature of their interaction between the device and the world external to the device.

Figure 2.2-1 SEPSIP Variables



2.2.1 Input Variables : Interacting with the Network

Input variables are those variables which interact with other devices through the network description. The network description consists of a number of nodes each having one or more subnodes. The node itself has no mathematical significance, it merely organizes subnodes into easily understood groups. The subnodes on the other hand, specify which network law should be applied to the input variables attached to it. Every input variable is assigned to one and only one subnode of a node. Each subnode however, can have an unlimited number of input variables assigned to it. There are four types of subnodes to which a variable can be connected to: voltage subnodes, reference voltage subnodes, current subnodes and reference current subnodes.

As an example, a node connecting a three phase motor to a transmission line would contain six subnodes: 3 voltage subnodes to relate each of the voltage phases and 3 current subnodes to relate each of the current phases.

2.2.1.1 Voltage Subnodes

All of the input variables attached to a voltage subnode are assigned the same value at all times. This voltage subnode value becomes one of the system variables that must be solved for. Although primarily used for communicating voltages, a voltage subnode can also be used to communicate other information between two devices. Examples include load sharing information and load shedding information.

2.2.1.2 Reference Voltage Subnodes

A reference voltage subnode is identical to a voltage subnode with the exception that the subnode voltage is specified as a fixed value and therefore is not a system variable. A reference voltage subnode must be used to set the ground potential, and may be used to set fixed operating points for certain elements.

2.2.1.3 Current Subnodes

A current subnode relates the variables attached to it by a conservation law which in electrical terms is known as Kirchhoff's Current Law. This law states that the sum of the variables attached to a current subnode is identically zero. In SEPSIP, this is accomplished by assigning the first variable attached to a current subnode the negative sum of the other attached variables. All of the input variables after the first one connected to the current subnode become system variables that must be solved for. The convention for current direction is that the current always enters the device and leaves a subnode.

2.2.1.4 Reference Current Subnodes

A reference current subnode does not satisfy Kirchhoff's Current Law. All of the input variables attached to it become system variables. In most simulations, Kirchhoff's Current Law at one subnode is a linear combination of all the Kirchhoff's Current Law equations from the other current subnodes. To specify the law again would result in either a system with too many implicit variables, or one which has a singular system Jacobian matrix.

2.2.2 Parameters

Parameters are variables that do not change through out a simulation. They are used when defining an element to customize a device model to fit the properties of a particular electrical component. Examples of parameters are resistances, inductances, capacitances, time constants, bias voltages, and saturation points.

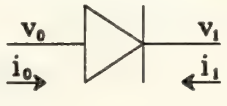
2.2.3 Implicit Definition of Input Variables

An important requirement for the equations describing a device is that the input variables must be implicitly defined. The network balancing algorithms determine the values the input variables take on. The device description provides information (feedback) as to how closely the implicit equations are satisfied through implicit variables. Implicit variables have a value of zero when their corresponding implicit equations are satisfied. One way to look at this process is to view a device as a transfer function between the input variables and the implicit variables. The Network then uses the implicit variables to iteratively generate the input variables until the implicit variables are driven to zero.

2.2.4 Implicit Variable Selection : Rotating and Translating Axes

One has a lot of latitude in defining the implicit variables. The easiest method is to write the constitutive equations, move everything over to one side, then define this quantity to be the implicit variable. This is the technique used in the last example which modelled the resistor. Unfortunately, this method can result in numerical instability when dealing with nonlinear devices.

Figure 2.2.4-1 Diode



Take a diode for example. A simple model for a diode is a switch that allows positive current to flow when the voltage across it 0.6 volts. Once current is flowing through the diode, the voltage drop across it is maintained at 0.6 volts. As with the resistor, the input variables are defined: v_0 and v_1 for the voltages, and i_0 and i_1 for the currents. The constitutive relations are:

$$\begin{aligned} i_0 = 0 \quad &: \quad v_0 - v_1 < 0.6 \\ v_0 - v_1 = 0.6 \quad &: \quad i_0 > 0 \end{aligned}$$

$$i_0 + i_1 = 0$$

The easy method of defining the constitutive equations would be to define I_0 and I_1 to be:

$$I_0 = \begin{Bmatrix} i_0 & : & v_0 - v_1 < 0.6 \\ v_0 - v_1 - 0.6 & : & v_0 - v_1 > 0.6 \end{Bmatrix}$$

$$I_1 = i_0 + i_1$$

This definition unfortunately, works badly in many circumstances. To begin with, the definition allows for negative current to flow when the diode is forward biased. Another problem is that the implicit variable I_0 is discontinuous at the boundary where v_0 - v_1 = 0.6. This type of discontinuity will usually cause much difficulty when trying to iteratively solve equations with most standard techniques (Such as the Newton Raphson method used in SEPSIP). In general, keeping the highest possible order derivative continuous across a boundary will help tremendously in achieving a numerically stable solution.

The definition of I_0 can be greatly improved by defining a new set of axes centered on the boundary point ($v = v_0 - v_1 = 0.6$ and $i_0 = 0$) and rotated 45 degrees. The transformation matrix to the new x and y variables is given by:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{\sqrt{2}}{2} \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} v_0 - v_1 - 0.6 \\ i_0 \end{pmatrix}$$

The constitutive equation for I_0 becomes:

$$y = |x|$$

I_0 can then be defined to be:

$$I_0 = y - |x|$$

This definition for I_0 is continuous. The discontinuity has been moved to the first derivative. For simple simulations where the voltage across the diode is not changing very rapidly compared to the simulation time increment, this definition for I_0 will normally work. Normally, one would like to have even higher order derivatives continuous to ensure numerical stability. This can only truly be done in this case by changing the constitutive equation to reflect more characteristics of a physical diode.

Even if the constitutive law of the diode is changed to make the slope continuous, the method outlined above for rotating axes should still be used. This is because most numerical methods rely on the partial derivatives of the implicit variables with respect to the input variables in the form of a Jacobian Matrix to update the last guess for the input variable. A very steep slope results in a Jacobian element being very large and the potential of having a floating point overflow when the Jacobian matrix is created or inverted. An overflow can also occur when the corrections to the input variables cause the recalculated implicit variable to overflow. In general, when the slope of the v/i characteristic has a section with a very steep slope, the axes should be rotated so that the maximum slope is minimized.

Another consideration when defining implicit variables is choosing the magnitude correctly. Since an exact solution which results in all the implicit variables being identically zero may not be possible due to time constraints or round off error, every iterative scheme relies on a method for determining when the implicit variables are close enough to zero. One way is to compare the root mean square of the implicit variables with a preset number. If this method is used, the order of magnitude of all the implicit variables should be the same for the same order of magnitude inputs. Otherwise, certain variables would be allowed to

vary more than other variables. For example, one voltage may be known to within a 1% error while another may be known to within a 0.1% error. Of course, if this effect is desired, one can easily weight an implicit variable by an arbitrary constant.

2.2.5 Data Storage : State Variables

A number of device models require the condition of the device during the last time increment be known. This information is conveyed to the model through state variables. Examples of states include the voltage and current of a capacitor or inductor, position of switches, position of breakers, time since a specific event occurred, and peak values of specific variables. One could conceivably use the state variables to store the time history of a variable to determine averages or other statistical or spectral properties.

2.2.6 External Inputs

External Inputs allow the user to interact with device models. The user can create a queue which contains the values an external input takes on at specified times. Uses for external inputs include position of switches, control waveforms, reference voltages, input waveforms from another program, controlling the configuration of an element.

2.2.7 External Outputs

External Outputs (along with External Inputs and Voltage Subnode voltages) are variables the user is allowed to monitor during the simulation. Therefore, any quantity that a user may be interested in should be defined as an external output variable. The user still has the choice as to which external outputs to see, so there is no problem with defining a number of output variables. External Output variables can also be stored in files for plotting at a later time.

2.2.8 Integration Techniques

Constitutive equations for devices often require the integration of a time derivative. Any textbook on numerical methods will provide a large selection of integration algorithms along with methods of determining their accuracy and stability. For most simulations

however, there are three methods that work well. The first is the Euler Forward method which is considered an explicit technique since it requires knowldege only of the values of the variables during the past time step.

$$\frac{dx}{dt} = y$$

$$x = x_{old} + y_{old} dt$$

The Euler Forward method is particularly suited for occasions when the differential equation has a strong mode that is much slower than the simulation time step. For systems with many modes, the Euler Forward method can eliminate the need to add additional implicit variables and associated input variables. The drawback of the Euler Forward method is that it requires small time steps for fast modes.

The Euler Backward method is similar to the Euler Forward method with the exception that the variables are evaluated at their present values instead of their old values. The Euler Backward Method is thus an implicit scheme since it uses present values to specify another present value. Since most devices will have several implicit variables to offset input variables, there is usually no extra computational burden in using an implicit scheme.

$$x = x_{old} + dt\, y$$

The Euler Backward method should normally be used where the possibility exists that the time increment will be longer than the time constant associated with any of the differential equations.

The implicitly defined Trapezoidal Method combines the Euler Forward and Backward methods:

$$x = x_{old} + \left(\frac{dt}{2}\right)(y + y_{old})$$

Whenever possible, one should use the trapezoidal rule due to its greater accuracy. However, when its use requires the addition of input variables to compensate for additional implicit variables, the Euler forward method should be considered.

There also exists a modified Trapezoidal method where the weights for **y** and $\mathbf{y}_{old}$ differ from 0.5. Weighting **y** slightly more helps prevent instabilities when the time step approaches the characteristic time constant of the equation.

2.2.9 Modelling Transfer Functions

Many devices have components that are modelled as transfer functions using Laplace Transforms. A common example has the form:

$$Y = \frac{As + B}{Cs + D} X$$

Since $\frac{dX}{dt} = Xs$, the Trapezoidal Method can be modified to provide an Implicit Variable

$$I_1 = A(x - x_{old}) + \left(\frac{dt}{2}\right) B(x + x_{old}) - C(y - y_{old}) - \left(\frac{dt}{2}\right) D(y + y_{old})$$

This equation can be incorporated in the definition of a more complicated device or can be defined seperately as its own device.

2.2.10 Jacobian Construction

SEPSIP uses a Jacobian Matrix to determine the corrections to the input variables in order to drive the implicit variables to zero. The elements of the Jacobian Matrix are the partial derivatives of the implicit variables with respect to all of the input variables. In other words, the Jacobian Matrix gives the slopes of the implicit surface in the directions of each of the input variables.

$$J = \begin{pmatrix} \frac{\partial I_1}{\partial x_1} & \frac{\partial I_1}{\partial x_2} & \frac{\partial I_1}{\partial x_3} & \cdots \\ \frac{\partial I_2}{\partial x_1} & \frac{\partial I_2}{\partial x_2} & \frac{\partial I_2}{\partial x_3} & \cdots \\ \frac{\partial I_3}{\partial x_1} & \frac{\partial I_3}{\partial x_2} & \frac{\partial I_3}{\partial x_3} & \cdots \\ \cdots & & & \end{pmatrix}$$

SEPSIP allows for the Jacobian to be calculated in two different methods. The device description can generate the Jacobian, or the network will approximate it. Normally the device description should create the Jacobian in the interest of speed and control over how it is created. It usually isn't too difficult to come up with analytic expressions describing the partials of the implicit variables. If desired, the network approximates the Jacobian by varying the input variables a small amount in either direction, and noting how much the implicit variables change. The change in the implicit variable divided by the change in an input variable is usually a fair approximation for the partial derivative.

Another advantage to having the device description generate the Jacobian is that the device description doesn't have to generate the real Jacobian matrix. If a value larger in magnitude than the partial derivative is substituted for an element of the Jacobian Matrix, the corrections to the input variables will result in the implicit variable being driven to zero more slowly. This can be advantageous near discontinuities of the constitutive equations and discontinuities in their first derivatives where one may want to retard the transition from one side of a discontinuity to the other.

Replacing an element of a Jacobian Matrix with a value smaller in magnitude than the partial derivative will usually result in a numerically unstable simulation. The corrections to the input variables will be larger than needed to drive the implicit variable to zero. The implicit variable will usually oscillate around zero and grow in magnitude with time.

2.3 ELEMENT DESCRIPTIONS

As described earlier, an element is a particular example of a device. A resistor for example, could be a device, while R11 which is a specific circuit element of type resistor having a resistance of 47 ohms would be an element. Elements are differentiated from each other by their names, device type, and parameter specification. The first section of the input file for SEPSIP contains all of the element definitions and parameter assignments.

2.4 NETWORK Description

The network is described by assigning all of the input variables from all of the elements to one and only one subnode. The subnodes in turn, are organized into groups called nodes. The purpose of the subnode is to relate the input variables of one element to the input variables of another variable. The nature of the relationship is determined by specifying the subnode to be either a voltage subnode or a current subnode. The relationship is further modified if the subnode is classified as a reference subnode.

Once the network is specified, system variables can be defined. System variables are members of the smallest subset of the input variables from which all of the other input variables can be derived from by using the relational properties of the subnodes. For a well defined simulation, the number of system variables will equal the total number of implicit variables. If the two numbers are not equal, there will either be many solutions to the simulation, or none at all. SEPSIP will not conduct a simulation unless the number of system variable does indeed equal the total number of implicit variables.

2.4.1 Voltage Subnodes

The input variables at a voltage subnode are all set equal to the subnode voltage which is a system variable (unless designated a reference subnode). As its name implies, a voltage subnode's purpose is to specify potential values. The potential value need not only be voltages however. Temperature, pressure, position, deflection, and Boolean states can also be communicated through voltage subnodes.

2.4.2 Current Subnodes

Input variables assigned to a current subnode satisfy Kirchhoff's Current Law (unless designated a reference subnode). The first input variable attached to the current subnode is set equal to the negative sum of all the remaining input variables attached to the subnode. All of the remaining input variables are system variables. The current subnode can therefore

be used where input variables must satisfy a conservation law through the network. Forces, torques, fluid flow rates and power flow can also be handled in addition to electrical current by the current subnode.

2.5 Reference Subnodes

To match the number of implicit variables with the number of system variables or to ensure there is a unique solution, it may be necessary to designate one or more subnodes a reference subnode. If a voltage subnode is specified to be a reference subnode, its value is always set to a user selected preset value. The normal usage would be to declare the ground voltage subnode to be a reference with zero value. Reference voltage subnodes can also be used to simulate power supply voltage busses.

Reference current subnodes do not satisfy Kirchhoff's current law. All of the input variables attached to them are designated system variables. In a closed system (i.e. one where the conservation law applies at every current subnode and in every element through out the system), Kirchhoff's current law at one of the current subnodes will be a linear combination of all the other Kirchhoff's current law equations at the other current subnodes. Therefore, the sum of the currents entering the reference subnode will automatically be zero.

Reference current subnodes also provide a way to leave input variables unterminated. This property can be used by device descriptions to increase the number of implicit variables used to represent the relations defining the device. Normally, when modelling electrical devices, one implicit equation should be provided for each terminal (which corresponds to two input variables : one voltage, and one current). This can be seen if we define:

m = number of terminals (voltage-current pairs) in the system
n = number of subnodes in the system
r_i = number of reference current subnodes
r_v = number of reference voltage subnodes
N_v = number of system variables due to voltage subnodes
N_i = number of system variables due to current subnodes
N_e = number of implicit equations

then

$$N_v = n - r_v$$

$$N_i = m - n + r_i$$

$$N_e = N_i + N_v = m - r_v + r_i$$

Since electrical devices deal with potential differences, rather than with the absolute magnitude of the potential, a reference potential is required to fix one of the nodes. As explained above, a reference current subnode is also required to prevent a singular system. The total number of implicit variables must therefore equal the number of terminals.

If a device description requires more implicit variables than it has terminals, extra input variables must be provided for the excess implicit variables. These variables should be attached to a reference current subnode or to separate voltage subnodes.

2.6 Conducting the Simulation

The simulation of the system described by the network description is carried out by solving a system of nonlinear equations at each time increment. Each nonlinear equation corresponds to the definition of an implicit variable which has a value of zero when the system is balanced. To balance the system, an initial guess is first made for all the system variables. For the first time step, the user may specify the guess, otherwise the system variables are all set to zero. For the remaining time steps, the results of the previous time step are used. The total number of independent system variables is considerably smaller than the total number of input variables since a number of the input variables are related through the network definition. For example, all the input variables attached to a voltage subnode are always given the same value. From the system variables, all of the input variables to each of the elements is derived from a description of the network topology. Using these values of the input variables, the implicit variables are calculated for each of the elements. If the mean square value of all the implicit variables are below a specified threshold, then the system is considered balanced. If the mean square value is larger than the threshold, then the system Jacobian matrix is constructed. The system Jacobian matrix consists of the partial derivatives

of the implicit variables with respect to each of the independent system variables. It is fabricated by piecing together the Jacobian matrices of all the elements. Inverting and multiplying the system Jacobian matrix by the vector containing the implicit variables provides a correction to the independent system variables. Once the corrections are subtracted from the independent system variables, the implicit variables are recalculated and the balancing process continues until the system is balanced. Once balanced, results are printed, state variables and external input variables are updated, and the time counter is incremented. In this manner, the simulation is stepped through time.

2.6.1 Setup

During the setup stage, two arrays of data structures are created to describe the network topology in a compact form. The first array describes all of the independent system variables and how they relate to the input variables of the individual elements. The second array keeps track of which implicit variable belongs to which element.

Each data structure for the system variable array consists of three subarrays. The number of entries in all three subarrays is equal to the number of input variables associated with the system variable. For a non-reference voltage subnode, all of the input variables attached to it will be associated with one system variable and will therefore each have entries in the three subarrays. A reference voltage subnode has a specified value and therefore is not associated with any of the system variables. For a reference current subnode, all of the input variables attached to it are separate system variables whose corresponding subarrays will contain only one entry. A non-reference current subnode's first input variable is set equal to the negative sum of the remaining input variables. Each of the remaining input variables is associated with one system variable whose subarrays have two elements: The first corresponding to the first input variable; and the second corresponding to the remaining input variable.

The first subarray for each system variable data structure contains integer offsets to the array of elements. The second subarray contains integer offsets to the input variable list within the element description. The third subarray contains a multiplier that is used in constructing the system Jacobian matrix. Normally this multiplier has a value of 1.0, but for the special case of a system variable corresponding to an input variable attached to a non-reference current subnode, the first entry has a value of -1.0 to account for the fact that the first variable associated with a current subnode is the negative sum of the remaining variables.

The data structures for the implicit variable structure array contain only two integer offsets. The first is an offset for the array of elements and the second is the offset in the array of implicit variables for the element. In this manner, all of the implicit variables can easily be referenced.

The setup section also creates an implicit variable cross-reference array in the description of each element that specifies which entry in the implicit variable structure array to which each of the implicit variables of the element corresponds.

2.6.2 Initialization

Before the simulation starts, all of the input variables, state variables, and external input variables are initialized. The initial values for the state variables are actually applied to the time increment immediately before the start of the simulation (old state variables). The input variables are initialized in a two step process. First, an array of system input variables is initialized. Then from the network description, the input variables for all of the elements are derived. If a variable is not explicitly initialized by the user, it is set to zero.

2.6.3 Updating External Inputs

The external input variables are updated at each time increment by scanning the external input queue for variable changes that occur before the present system time. All of the external input variables scheduled for a change in value are then updated.

2.6.4 Balancing the System

The process of finding a set of input variables that simultaneously satisfies all the network equations and all the implicit equations of the elements is known as **balancing the system**. The procedure involves calculating the implicit variables, constructing a system Jacobian matrix, calculating corrections to the input variables, and repeating the process until the implicit variables are within tolerable limits of zero.

2.6.4.1 Calculating Implicit Variables

The first step in balancing the system is calculating the implicit variables. The functions describing each of the elements are called one at a time and provided with the appropriate input variables. These functions use the input variables along with the external input variables and the state variables calculated in the previous time step to generate the implicit variables.

Once all of the implicit variables have been calculated, they are assembled into an implict variable array in the order specified in the implicit variable structure array constructed in the setup phase. The mean square value of all the implicit variables is also calculated and if its magnitude is smaller than a predefined amount, the sytem is considered balanced and the program jumps to printing the results out.

2.6.4.2 Manufacturing System Jacobian Matrix

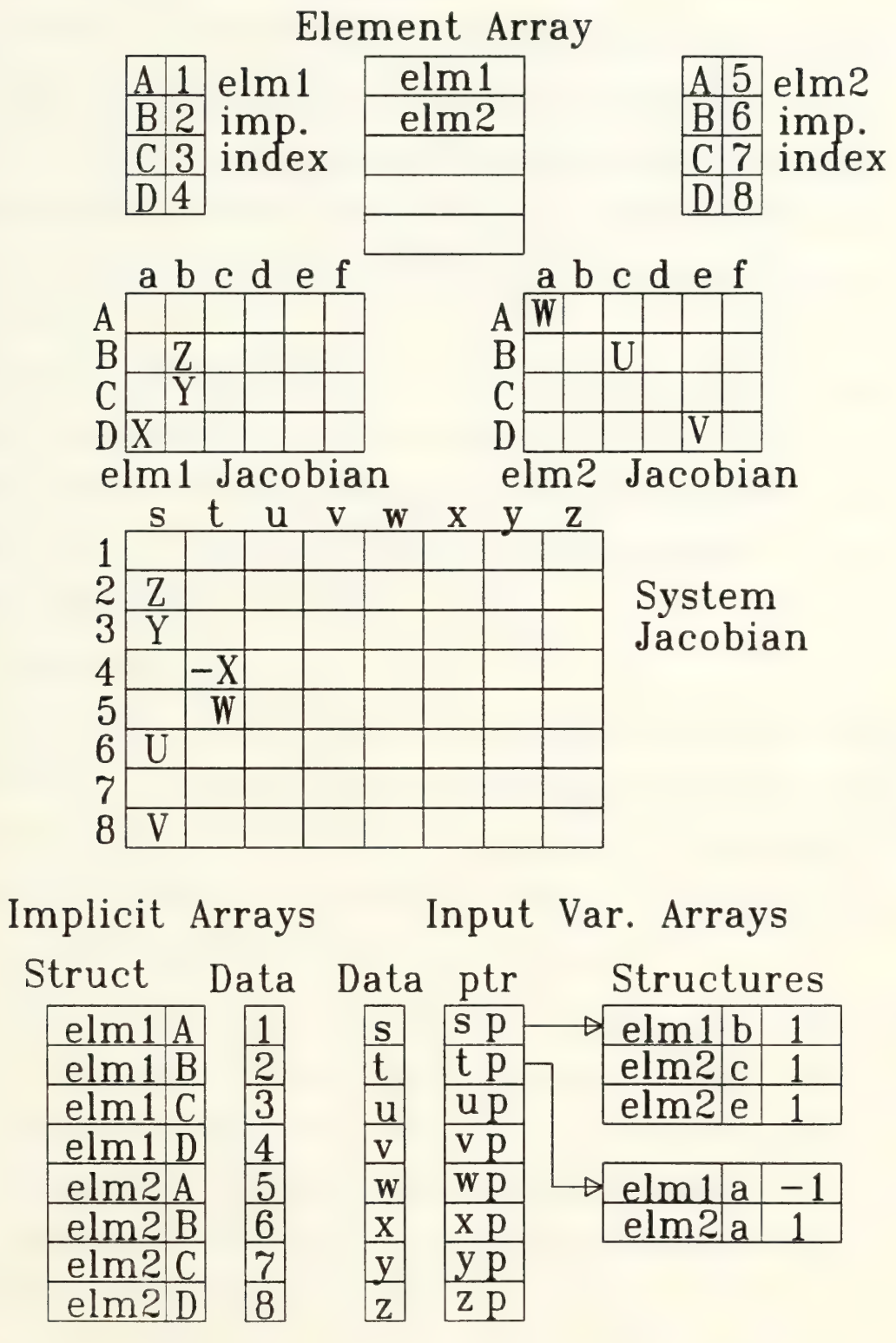
The system Jacobian matrix is generated by piecing together the Jacobian matrices for each of the elements. The elemental Jacobian matrix can be generated by the function which also produces the implicit variables, or it can be approximated numerically. The function that returns the implicit variables also returns a flag indicating whether or not the Jacobian matrix was calculated. If the matrix was not constructed, it is manufactured by varying each of the input variables slightly and approximating the partial derivatives by

dividing the differences between the resulting implicit variables by the differences of the input variables. The percentage change and the minimum change in the input variables can be specified by the operator.

The system Jacobian Matrix is constructed one column at a time. Each column has its associated structure in the input variable structure array that specifies which elements and input variables contribute to that column. Knowing the element and the input variable is enough knowledge to extract the appropriate column from the element Jacobian matrix. Which row in the system Jacobian matrix to insert each of the entries of the element Jacobian column is provided by the implicit variable cross reference array. By stepping through each of the structures of the input variable structure array, the entire system Jacobian matrix can be constructed.

One result of separating the creation of the system Jacobian matrix from the element Jacobian matrices is that extra work is done in creating columns in the element Jacobian matrices that do not contribute anything to the System Jacobian. This arises whenever an input variable is attached to a reference voltage subnode. Since a reference voltage node always has a constant voltage, it does not contribute a system variable.

Figure 2.6.4.2-1



2.6.4.3 Calculating Variable Corrections via Newton-Raphson

Once the system Jacobian matrix $\bar{J}$ and the implicit variable vector $\bar{i}$ are created, corrections to the input variable vector $\bar{x}$ can be calculated via the Newton-Raphson method. The matrix equation $\bar{J}\bar{x} = \bar{I}$ is solved using Gaussian Elimination with partial pivoting. If the Jacobian Matrix is singular, the Gaussian Elimination will fail due to the inability to get a non-zero number in the pivot element. If this occurs, the simulation halts with an error message.

After the corrections have been applied to the input variable vector, the implicit variables are recalculated and the process continues until the mean error of the implict variables is within tolerable limits of zero, or until a predetermined number of iterations have been made. If the iteration limit is reached, the simulation has failed to converge on a solution is halted with an error message.

2.6.5 Printing Results

Since the operator can specify a printing time increment different from the simulation time increment, a test is made to determine whether or not any results should be printed. If the test is successful, all of the variables designated to be displayed in the simulation section of the input file are printed to the screen, or to a file if one was specified by the operator.

2.6.6 Updating State Variables and Time Counter

The state variables are produced by the function that also calculates the implicit variables. Once the system is balanced, these state variables are moved to another array called the **old state variable array** which can be used during the next time increment.

The time variable is also updated by adding the specified time increment. If the time variable exceeds the maximum time of the simulation, the simulation is terminated and control returns back to the main menu of SEPSIP.

2.6.7 Potential Problems

2.6.7.1 Numerical Instability

The Newton-Raphson method is only guaranteed to converge on a solution if the initial guess is sufficiently close to the solution. Unfortunately, it is very difficult to determine how close, 'sufficiently close' is. If the time increment is small enough, the input variables should not change appreciably. Hence using the results of the previous time increment as a first guess usually produces good results. There are two occasions however, when this may not hold. First, during the initial balancing of the system, the initial guesses are provided by the user. If these guesses are not sufficiently close to the solution, the system will not converge. Another situation that may occur during the execution of a simulation is that a discrete event may occur that changes the configuration. The solution to the new configuration may not be sufficiently close to the solution of the old configuration to guarantee stability.

2.6.7.2 Singular Jacobian Matrix

The simulation can also fail if the System Jacobian Matrix is singular and therefore uninvertable. This can occur if the network is defined poorly or if a discrete event results in a poorly defined network. Systems incorporating switches or breakers are particularly susceptible to this problem. (If two switches are connected in series and both opened, their implicit variables would be set equal to the terminal currents. The current subnode connecting the two switches would further equate the two attached currents and thereby overspecify them. Furthermore, the voltage of the connecting voltage subnode would not be implicitly defined anywhere.)

2.6.7.3 Non-Unique Solutions

In nonlinear systems, it is often possible for more than one set of input variables to satisfy all of the constitutive relations and network equations. For these systems, it is very important to provide the solution with the best possible intitial guesses in order for the

system to converge on the desired solution. Once the simulation has started, using the results of the previous time step as an initial guess should normally result in convergence to the proper solution. This method for determing the initial input variables can still fail during time steps in which system reconfigurations have taken place that result in certain variables changing considerably over the one time increment.

CHAPTER 3

SEPSIP

SHIPBOARD ELECTRICAL PLANT

SIMULATION PROGRAM

3.1 Introduction

SEPSIP is a simulation program optimized for solving lumped parameter systems with elements that are described by nonlinear constitutive equations. The program is written in the C programming language and is presently running under the UNIX operating system on Digital Equipment Corporation VAX Workstation II and VAX Workstation 2000 computers. The files are located in the **sepsip** subdirectory of the 13.411 Course Locker of MIT's Project ATHENA. SEPSIP should be easily adapted to other computers and operating systems since a minimum of machine specific routines have been used.

Running a simulation with SEPSIP is a three stage process. First, an input file must be created with a text editor such as EMACS. The simulation is then carried out by the SEPSIP program with the results printed to an output file. Finally, the output file is printed directly out or sent to a plotting program such as NORPLOT for viewing.

This chapter describes how to create an input file and lists the commands available when executing SEPSIP. Actual examples of input files can be found in Chapter 5.

3.2 Data Entry Conventions

Much effort has been made to ease the task of creating the input files for SEPSIP. The input files are very loosely structured in the sense that data need not be entered in specific columns, comments can be inserted anywhere, other files can be referenced through "include" statements, and to a certain degree, the order of lines is not rigid.

3.2.1 Acceptable Characters

Virtually all of the printing ASCII characters can be used in naming elements, variables, nodes and subnodes. To prevent confusing the program, in addition to "white" characters (spaces, tabs, newlines) the following characters should be avoided entirely:

: =

Additionally, the following characters should not be used for the first character:

! # + - 0 1 2 3 4 5 6 7 8 9

These characters should not be used for the last character:

. \

3.2.2 Reserved Names

The following keywords should not be used for naming elements, variables, nodes or subnodes:

end
external
include
initial
node
simulation

All other keywords can be used, but for clarity, should be avoided.

3.2.3 Specifying Variables and Subnodes

Variables are specified in the following format:

element : variable

The colon is used to delimit the element name from the variable name. Tabs and / or spaces between the element name and the colon and between the variable name and colon are optional.

Subnodes are similarly described:

node : subnode

3.2.4 Numerical Entries

The following formats for entering numbers are valid:

123	**integer**
-123.456	**floating point**
123e-4	**exponential**
+123E3	**exponential**

3.2.5 White Characters

Spaces and tabs are used to separate data elements. Any number and combination of spaces and tabs may be used. A data line is terminated with a "newline" character (also known as a carriage return).

3.2.6 Continuation Lines

In general, each line of the input file must be shorter than 80 characters. This usually is not a problem since there isn't very much information that must be included on one data line. Continuation lines are allowed however, in the Network Description section. This section requires the grouping of a number of variables together. It is therefore quite likely that more than 80 characters would be required. Consequently, for this section alone, a line can be terminated with \ or ... to indicate the data continues on the following line.

3.2.7 Case Sensitivity

Keywords are case insensitive (Both upper and lower case letters accepted), all other entries are case sensitive.

3.2.8 Comment Lines

Any Line beginning with a **!** or **#** is ignored. Hence comment lines can be inserted anywhere within the file by preceding them with **!** or **#**. Blank lines are also ignored.

3.3 Input File Generation

SEPSIP requires an input file to describe the simulation. This file consists of four sections: Element Description, Network Description, Initialization, and Simulation Description. The input file can be created or edited by any text editor.

Figure 3.3-1 Sample SEPSIP Input File

```
! SEPSIP Input File
!
! Element Description
device_1  elm_1a
  par_1 1.0
  par_2  3
  end
device_2  elm_2
  par_1  2.0e-6
  end
! Network Description
NETWORK
!
NODE gnd
  rv:volt = 0.0 = elm_1a:v0 = elm_2:v0
  ri:current    = elm_1a:i0 = elm_2:i0
  end
NODE A
  v: volt = elm_1a:v1 = elm_2:v1
  i: current = elm_1a:i1 = elm_2:i1
  end
! Initialization Section
INITIALIZE
  elm_1a : state_1 37
  elm_1a : v1      32.2
  end
NODE VOLTAGE INITIALIZATION
  A:volt 100.0
  end
EXTERNAL INPUTS INITIALIZATION
  elm_2 : ext_in  24
  end
!Simulation Section
SIMULATION
Display
  elm_1a : ext_out_1
  A : volt
  end
TIME_STEP   1.0e-3
PRINT_STEP  5.0e-3
TMIN        0.0
TMAX        1.0
```

3.3.1 Organization -- Using INCLUDE Files

A common problem among many simulation programs is the requirement that all the necessary information be contained in one file. For large simulations, this results in long input files that are difficult to manage and edit. SEPSIP addresses this problem with the **include** keyword. Outside any data block (A data block begins with a keyword and concludes with the **end** statement) the keyword **include** followed by a filename results in the insertion of the contents of the 'included' file at the location of the **include** keyword. 'Included' files may also contain **include** keywords. This feature allows one to organize the input file in a number of ways. Figure 3.3.1-1 shows one method of using the **include** keyword.

Figure 3.3.1-1 Using the INCLUDE Keyword

```
! t.all
! t.elm contains the element descriptions
include t.elm
! t.net contains the network description
include t.net
! t.init contains the initialization section
include t.init
! t.sim contains the simulation section
include t.sim
```

3.3.2 ELEMENT Description

The first section of the SEPSIP input file is the ELEMENT Description. This section defines the elements and specifies all of the parameters for the elements. A data block for defining an element has the following format:

Figure 3.3.2-1 ELEMENT Description

```
element_name device_name
  parameter  value
  parameter  value
  parameter  value
  end
```

Element_name can be any single word as long as it conforms to the conventions of Section 3.2. **Device_name** is the name of the particular device. A list of available devices

can be obtained by running SEPSIP and entering **dd** at the first prompt. To obtain more information on a particular device (including a list of **parameters**) enter **dD device_name**. A listing of all the device descriptions can be written to a file by entering **dw filename**.

When defining an element, all of the parameters must be specified. An error is generated whenever an **end** statement is reached and all of the parameters have not been provided with values.

Elements can be defined that are not used in the network description. This allows for the creation of a 'junk box' of parts that can be used when building and modifying the network description. A warning will be generated when an element is defined but not used. It is also a good idea not to have too many 'spare elements' since execution time will slow down somewhat.

The Element Description section ends when the keyword **network** is encountered.

3.3.3 NETWORK Description

The keyword **network** signals the beginning of the network description. This section consists of data blocks that describe each of the network nodes. All of the lines within the data block (except the first and last) describe one subnode. Each data block has the format:

Figure 3.3.3-1 NETWORK Description

```
NODE node_name
 subnode_ind : subnode_name = elm : var = elm : var
 subnode_ind : subnode_name = elm : var = elm : var
 END
```

Node_name and **subnode_name** once again, can be any word following the conventions of section 3.2. **Node_name** must also be distinct from any of the element names as well. **Subnode_ind** specifies the type of subnode and consists of up to three characters, two of which are optional. The format of the **subnode_ind** is:

Reference Indicator: [optional] An **r** as the first character of **subnode_ind** specifies that subnode to be a reference subnode.

Subnode Type Indicator : [mandatory] The next character must be either an **i** or a **v** to specify the subnode as either a current or voltage subnode.

Grouping Indicator : [optional] **Subnode_ind** can end with a digit greater than zero to specify the number of consecutive subnodes that should be created. If this digit is greater than one, then that number of consecutive subnodes are created. The first subnode will have **subnode_name** as its name and include all of the specified variables. The following subnodes will use **subnode_name** appended by **_b** , **_c** , etc. and use the next consecutive input variable for each of the elements. This feature allows one to connect together 'multiple conductor cables' with one single entry. Typically, this will be a **3** for three phase systems.

Subnodes can also be designated a reference subnode in the simulation section of the input file. For clarity it is better to define all of the reference subnodes in the NETWORK section.

In the special case of a **Reference Voltage Subnode**, the reference voltage may be specified immediately following the subnode name as demonstrated in figure 3.3.3-2. This reference voltage value however, can be overwritten by an entry in the **REFERENCE** block of the **SIMULATION** section of the input file. If a reference voltage subnode's voltage is not specified either in the NETWORK or the SIMULATION section, it is set to a value of zero.

Figure 3.3.3-2 NETWORK Description Example

```
!
! t.net
! Norbert H. Doerry   12 March 1989
!
NETWORK
! The following node has both reference voltage
! and current subnodes
!
NODE gnd
    rv:v = gen:v0n = load:v0n = meter:v0
    ri:i = gen:i0n = load:i0n
    end
!
! The next node shows how to specify each phase
! independently
!
NODE A
    v:v_a = gen:v0a = sw:v0a = meter:v1
    v:v_b = gen:v0b = sw:v0b
    v:v_c = gen:v0c = sw:v0c
    i:i_a = gen:i0a = sw:i0a
    i:i_b = gen:i0b = sw:i0b
    i:i_c = gen:i0c = sw:i0c
    end
!
! The next node shows how to use the grouping indicator
!
NODE B
    v3:v = sw:v1a = load:v0a
    i3:i = sw:i1a = load:i0a
    end
!
! The next node shows how to use reference voltages to set
! operating points
!
NODE GEN_REFS
    rv:freq = 60.0 = gen:freq
    rv:Vmag = 100.0 = gen:Vmag
    end
```

3.3.4 INITIALIZATION Description

The Initialization Section is the only optional sectional in the input file. If a variable is not explicitly initialized, its value is set to zero. Therefore, one only needs to initialize the non-zero variables. The following types of variables may be initialized:

Input Variables attached to Current Nodes
Node Voltages
State Variables
External Input Variables

Input Variables attached to Voltage Nodes may also be initialized, but the Node Voltage initialization will take precedence and overwrite the Input Variable initialization.

The Initialization Section is composed of three subsections that may be entered in any order, or omitted if not used. These three subsections are: **INITIALIZE, EXTERNAL [INPUTS INITIALIZATION], NODE VOLTAGE [INITIALIZATION].**

3.3.4.1 INITIALIZE

The **INITIALIZE** subsection is used to initialize input and state variables for any element. The format for this subsection is:

Figure 3.3.4.1-1 INITIALIZE subsection

```
INITIALIZE
    element_name : variable_name  value
    element_name : variable_name  value
    element_name : variable_name  value
    end
```

Variable_name can be the name of either a state variable or the name of an input variable. For state variables, **value** becomes the **old state variable** for the first time increment. For input variables, **value** is the first guess used for input variables attached to current subnodes. If an input variable is attached to a voltage subnode, **value** is ignored.

The INITIALIZE subsection ends when the keyword **end** is encountered.

3.3.4.2 EXTERNAL [INPUTS INITIALIZATION]

The **External Inputs Initialization** subsection begins with either the keywords **EXTERNAL** or **EXTERNAL INPUTS INITIALIZATION**. Its purpose is to provide the default values for the external input variables. The default values set in this subsection can be overwritten by entries in the **EXTERNAL INPUTS** subsection of the **SIMULATION** section.

Figure 3.3.4.2-1 EXTERNAL INPUTS INITIALIZATION Subsection

```
EXTERNAL INPUTS INITIALIZATION
    element_name : external_input_name  value
    element_name : external_input_name  value
    element_name : external_input_name  value
    end
```

3.3.4.3 NODE VOLTAGE [INITIALIZATION]

The **Node Voltage Initialization** subsection begins with either the keywords **NODE VOLTAGE** or **NODE VOLTAGE INITIALIZATION**. Its purpose is to provide the initial guesses for all the input variables attached to a voltage subnode. The subsection ends when the keyword **END** is encountered.

Figure 3.3.4.3-1 NODE VOLTAGE INITIALIZATION Subsection

```
NODE VOLTAGE INITIALIZATION
    node_name : subnode_name  value
    node_name : subnode_name  value
    node_name : subnode_name  value
    end
```

3.3.5 SIMULATION Description

The **SIMULATION** section begins with the keyword **SIMULATION** and continues until the end of the input file is reached. The Simulation section details the manner in which a simulation is carried out. The following keywords can be included in the simulation section:

CONVERGE
DELTA
DELTA_MIN
DISPLAY
EXTERNAL INPUTS
MAX_ITERATION
PRINT_STEP
REFERENCE
TIME_STEP
TMAX
TMIN

3.3.5.1 CONVERGE

Format: **CONVERGE value**

Value is the maximum mean square error of all the implicit variables allowed for a balanced system. Note that since **CONVERGE** is applied to the average of the implicit variables, any single implicit variable may have a square magnitude considerably larger than **value**.

3.3.5.2 DELTA

Format: **DELTA value**

Value is the fractional amount an input variable is changed when the network calculates the Jacobian matrix of an element using the secant method. The input variable are multiplied by **(1 + value)** and **(1 - value)** and if the difference between the two resulting numbers is greater than twice **DELTA_MIN**, they are used to recalculate the implicit variables. The differences between the implicit variables divided by the differences between the two values of the input variables provide the column of the element Jacobian matrix corresponding to that input variable. **DELTA** is only significant if at least one of the elements used does not calculate the Jacobian matrix within its defining function.

3.3.5.3 DELTA_MIN

Format: **DELTA_MIN value**

DELTA_MIN is used in conjunction with **DELTA**. If when calculating an element Jacobian matrix by the secant method, the difference between the two offset input variables is greater than twice **DELTA_MIN**, then **DELTA_MIN** is added and subtracted from the input variable for the purpose of calculating the partial derivative.

3.3.5.4 DISPLAY

Format:

```
DISPLAY
 element_name : external_output_variable_name
 element_name : external_output_variable_name
 node_name : voltage_subnode_name
 node_name : voltage_subnode_name
 end
```

DISPLAY specifies which variables are written to the screen or to the specified file when the simulation is conducted. Only **external output variables** and **voltage subnodes** can be displayed. **PRINT_STEP** specifies how often the variables are displayed.

3.3.5.5 EXTERNAL INPUTS

Format:

EXTERNAL INPUT
element_name : external_input_variable_name value time
element_name : external_input_variable_name value time
element_name : external_input_variable_name value time
end

EXTERNAL INPUT provides the information needed to produce an external input queue that tells the simulation when the value of an external input value should be changed. **Time** is the simulation time at which the specified external input variable should be set to **value**.

3.3.5.6 MAX_ITERATION

Format: **MAX_ITERATION value**

Value is the maximum number of iterations that are performed during any single time interval in an attempt to balance the system. If the system can not be balanced in fewer iterations, an error message is printed and the simulation is halted.

3.3.5.7 PRINT_STEP

Format: **PRINT_STEP value**

Value specifies how often the variables listed in **DISPLAY** are printed.

3.3.5.8 REFERENCE

Format:

REFERENCE
v : node_name : voltage_subnode_name value
v : node_name : voltage_subnode_name value
i : node_name : current_subnode_name
end

The REFERENCE subsection can declare subnodes defined in the NETWORK section to be reference subnodes (whether or not they were defined previously to be ref-

erence subnodes in the **NETWORK** section). Since the Simulation section may be modified after an input file has been loaded into SEPSIP, this section can also be used to vary the voltage of a reference voltage subnode between simulations. **Value** overrides the default value provided in the **NETWORK** description section.

3.3.5.9 TIME_STEP

Format: **TIME_STEP value**

Value is the time step used in calculating the simulation.

3.3.5.10 TMIN

Format: **TMIN value**

Value is the initial value that the time counter is initialized to. After each time the system is balanced, the time counter is incremented by the **TIME_STEP**.

3.3.5.11 TMAX

Format: **TMAX value**

Value is the largest value that the time counter can take on. If the time counter exceeds **value**, the simulation is successfully concluded and control passes back to the main menu of SEPSIP.

3.4 Running the Simulation

3.4.1 Starting SEPSIP

The method for executing the SEPSIP program depends on the operating system being used. On MIT's Project ATHENA, the following procedures should be used:

athena% **attach 13.411**
athena% **/mit/13.411/sepsip/sepsip**

or

athena% **attach 13.411**
athena% **/mit/13.411/sepsip/sepsip input_filename**

The program starts by printing a welcome message followed by the version number and date. If **input_filename** is specified, it is loaded. Any errors detected are listed as well as the opening of any include files. SEPSIP then enters the main menu and prompts for the first command.

3.4.2 Command Entry Conventions

3.4.2.1 SEPSIP Menus

SEPSIP is a menu oriented program consisting of one main menu and several sub-menus. The menus are organized in two columns: The first contains single characters used to execute the commands listed in the second column. After the menu is displayed, the user can enter the character corresponding to the desired command followed by a carriage return.

Several of the menus will have a variable number of options depending on the state of the simulation. If a valid input file has not been loaded for example, the main menu will not have the **Conduct Simulation** or **Continue** commands available since they would be meaningless.

The **Conduct Simulation** (option **s**) and **Continue** (option **c**) commands from the main menu can be followed by an output filename. If a filename is specified, output from the simulation is redirected from the screen to the specified file.

3.4.2.2 Concatenating Commands

SEPSIP allows one to execute an option in a submenu directly from the main menu by entering the character that executes the submenu followed by the desired character from that submenu. These two characters can be separated by spaces or tabs and can also be followed by whatever input text is required by the selected command.

Examples:

dd	(displays device summary)
fc /mit/yourname	(changes the working directory)
u p t.plot	(executes plotting program with argument t.plot)

3.4.2.3 Input Filename Specification

In several of the options, the user is prompted for an input filename. Any existing file can be entered, including a path specification if required. If a default filename is offered, a carriage return will select the default. If no default filename is presented, a carriage return will terminate the command. Should SEPSIP be unable to open the file, another filename is prompted for. Entering **?** as a filename results in the listing of the current directory. A **q** terminates the command.

3.4.2.4 Output Filename Specification

An Output filename can be specified for several commands. Any filename recognized as legal by the operating system may be used. If a default filename is offered, a carriage return will select the default. If no default filename is presented, or if the filename **stdout** is entered, a carriage return will result in the file being listed on the screen. If for some reason, SEPSIP is unable to open the file, another filename is prompted for. Entering **?** as a filename results in the listing of the current directory. A **q** terminates the command.

3.4.3 Command Summary

This section lists all the commands available in SEPSIP. A more detailed explanation of the commands is provided in the following sections.

c [file]	**Continue simulation**
d	**Switch to Display Data menu**
dc [dir]	**Change Working Directory**
dd	**Display Device Summary**
dD	**Display Device Data**
de	**Display Element Summary**
dE	**Display Element Data**
dn	**Display Network Summary**
dq	**Quit Display Data menu**
dw [file]	**Write Device Data File**
e	**Switch to Edit Simulation Parameters Menu**
ed	**Switch to Edit Display Variable list Menu**
eda	**Add Variable to Display Variable list**
edd	**Delete Variable from Display Variable list**
edq	**Quit Edit Display Variable list Menu**
ej	**Edit Jacobian Parameters**
eq	**Quit Edit Simulation Parameters Menu**
er	**Edit Reference Voltage Subnode Voltages**
et	**Edit Time Parameters**
f	**Switch to File Options Menu**
fd [file]	**Dump Simulation State**
fi [file]	**Save INITIALIZATION Section**
fI [file]	**Load INITIALIZATION Section**
fq	**Quit File Options Menu**
fs [file]	**Save SIMULATION Section**
fS [file]	**Load SIMULATION Section**
q	**Quit : Terminate SEPSIP program**
s [file]	**Conduct Simulation**
u	**Switch to Utility Menu**
ue [file]	**Execute EMACS text editor**
up [file]	**Execute Norplot Plotting Package**
u?	**Display Directory**
u% [cmd]	**Execute System Command**
u+	**Perform Screendump to the default printer**

3.4.4 Main Menu

This section describes all the commands available in SEPSIP. For each command, the format for executing it from the main menu is presented along with a description of the command. For executing a command within a submenu, the first letter should be omitted.

3.4.4.1 c Continue

Format:

c

c filename

The **Continue** command is only available if a valid input file has been loaded and a simulation has already been conducted. Its purpose is to allow the simulation to continue without reinitializing any of the variables. To use this command though, **tmax** must be changed to a higher value that corresponds to the new desired ending time.

If **Continue** is invoked without a filename, the results of the simulation are displayed on the screen, otherwise the results are written to the specified file.

==========

3.4.4.2 d Display Data

Format:

d

d command

Display Data presents a submenu with the following options:

==========

3.4.4.2.1 d c Change Working Directory

Format:

dc

dc directory

This command changes the working directory for specifying both input and output files. If a directory name is not specified on the command line, the user is prompted for one. For systems operating under the UNIX operating system, this is the only method available since a **cd** command executed as a system call will not work.[1]

==========

3.4.4.2.2 d d Display Device Summary

Format:

dd

Display Device Summary lists the names of all the available devices.

==========

---

1 Under UNIX, when a system call is made from a program, a new shell is created for the specified command to be executed in. When the command terminates, the shell disappears. Therefore, if a **cd** command is executed, it will change the directory in the new shell and then terminate. The new shell will immediately disappear and control will pass back to the old shell whose working directory was never altered.

3.4.4.2.3 d D Display Device Data

Format:

dD

dD device_name

Display Device Data provides detailed information about a particular device. If **device_name** is not specified, it will be prompted for. All of the variable names associated with **device_name** are listed.

==========

3.4.4.2.4 d e Display Element Summary

Format:

de

Display Element Summary is only available if a valid input file has been loaded. All of the defined elements are listed along with which devices they are associated with. If an element is not used in the network description, its entry is appended with

"***** Not Used *****".

==========

3.4.4.2.5 d E Display Element Data

Format:

dE

dE element_name

Display Element Data is only available if a valid input file has been loaded. If **element_name** is not specified, it is prompted for. All of the variables associated with **element_name** and their values are listed.

==========

3.4.4.2.6 d n Display Network Summary

Format:

dn

Display Network Summary is only available if a valid input file has been loaded. For each node, the constitutive subnodes and their attached variables are displayed. After all the data for a node has been presented, the user is prompted to enter a carriage return to continue. If a **q** is entered instead, the command is terminated. A **b** will result in the previous node being listed.

==========

3.4.4.2.7 d q Quit

Format:

dq

Quit returns control back to the main menu.

==========

3.4.4.2.8 d w Write Device Data File

Format:

dw

dw device_data_filename

Write Device Data File prints out all of the device data for all of the devices. If **device_data_filename** is not specified, the user is prompted for it. For a particular device, this command presents all the same information as **Display Device Data**

==========

3.4.4.3 e Edit Simulation Parameters

Format:

e

e command

Edit Simulation Parameters presents a submenu for editing data from the simulation section of the input file. This command is only available if a valid input file has been loaded.

==========

3.4.4.3.1 e d Edit Display Variables

Format:

ed

ed command

Edit Display Variables presents a submenu for adding and subtracting variables from the display variable list. In addition to the submenu, all of the variables presently on the list are displayed. The values of the variables on this list are displayed during the simulation in increments of **print_step** as set in the input file or by **Edit Time Parameters.**

==========

3.4.4.3.1.1 a Add Display Variable

Format:

eda

eda element_name : external_output_variable

eda element_name : external_input_variable

eda node_name : voltage_subnode_name

Add Display Variable adds a variable to the display variable list. External Input, External Output, and Voltage Subnodes may all be specified.

==========

3.4.4.3.1.2 d Delete Display Variable

Format:

edd

edd element_name : external_output_variable

edd element_name : external_input_variable

edd node_name : voltage_subnode_name

Delete Display Variable deletes a variable presently on the display variable list.

==========

3.4.4.3.1.3 q Quit

Format:

edq

Quit returns to the **Edit Simulation Parameters** submenu

==========

3.4.4.3.2 e j Edit Jacobian Parameters

Format:

edj

Edit Jacobian Parameters allows the user to change the following simulation parameters:

CONVERGE

MAX_ITERATION

DELTA

DELTA_MIN

The user is prompted to enter a new value for each of these parameters. If a carriage return alone is entered, the default value is used. If a **q** is entered, the command is terminated. A **b** allows the previous variable to be changed.

==========

3.4.4.3.3 e q Quit

Format:

eq

Quit returns control back to the main menu.

==========

3.4.4.3.4 e r Edit Reference Voltage Subnode

Format:

er

er node_name : reference_voltage_subnode_name

er node_name : reference_voltage_subnode_name value

Edit Reference Voltage Subnode allows the user to change the value a reference voltage subnode is set to. If the command is executed without specifying the subnode, a list of the reference voltage subnodes is provided before the user is prompted for the subnode name.

==========

3.4.4.3.5 e t Edit Time Parameters

Format:

et

Edit Time Parameters allows the user to change the following simulation parameters:

TIME_STEP

TMIN

TMAX

PRINT_STEP

The user is prompted to enter a new value for each of these parameters. If a carriage return alone is entered, the default value is used. If a **q** is entered, the command is terminated. A **b** allows the previous variable to be changed.

==========

3.4.4.4 f File Options

Format:

f

f command

File Options presents a submenu for reading and writing several different types of files.

==========

3.4.4.4.1 f d Dump Simulation State

Format:

fd

fd filename

Dump Simulation State prints to a file, the entire state of the simulation. Every variable for every element is listed along with the system Jacobian matrix and associated variables. While the file produced by this command may become very large, it is often the only way to find the cause of a simulation's failure to converge.

==========

3.4.4.4.2 f i Save INITIALIZATION Section

Format:

fi

fi filename

Save INITIALIZATION Section writes to a file, all of the current values of the **input variables**, **state variables**, **external input variables**, and **voltage subnode voltages**. This file can then be included in another input file to specify a starting point for further simulations. This command allows one to run a simulation until steady state has been achieved, save the initialization section, then conduct simulations to study the effects of a disturbance on the steady state solution.

==========

3.4.4.4.3 f I Load INITIALIZATION Section

Format:

fI

fI filename

Load INITIALIZATION Section loads from a file, the initial values of the **input variables**, **state variables**, **external input variables**, and **voltage subnode voltages**. The file must conform to the format specified in section 3.3.4. The easiest way to create this file is to use or edit a file created by the **Save INITIALIZATION Section** command.

==========

3.4.4.4.4 f s Save SIMULATION Section

Format:

fs

fs filename

Save SIMULATION Section writes to a file, the **SIMULATION Description** section of the input file as described in section 3.3.5. If **stdout** is used as a filename, the **SIMULATION Description** section is listed on the screen. This is a fast way of seeing all the simulation variables at once.

==========

3.4.4.4.5 f S Load SIMULATION Section

Format:

fS

fS filename

Load SIMULATION Section reads from a file, the **SIMULATION Description** section of the input file as described in section 3.3.5. Files created with the **Save SIMULATION Section** command can be directly loaded with this command.

==========

3.4.4.5 q Quit

Format:

q

Quit results in the termination of SEPSIP. Control is passed back to the operating system.

==========

3.4.4.6 s Conduct Simulation

Format:

s

s filename

Conduct Simulation starts the simulation. If a filename is not specified, the values of all of the variables on the **display variables** list are printed to the screen. If a filename is specified, the values of the variables are printed to the designated file. A period (.) is printed on the screen every time a line is printed to the file. This allows one to see that the simulation is actually proceeding and the program is not stuck in an infinite loop.

==========

3.4.4.7 u Utilities

Format:

u

u command

Utilities presents a submenu that is fundamentally different from the other menus in that it is entirely user defined. The file **sepsip_util.menu** contains all the data required to create and execute a menu. Appendix D describes how to edit this file to add or delete menu items. The following commands are presently implemented on MIT's Project ATHENA.

==========

3.4.4.7.1 u e Editor -> emacs

Format:

ue

ue filename

This command executes the **emacs** text editor.

==========

3.4.4.7.2 u p Plotting -> Norplot

Format:

up

up filename

Norplot is a simple X-Window oriented plotting package that allows for the input file to have multiple columns of data.

==========

3.4.4.7.3 u ? List Directory

Format:

u?

u? directory_path

This command lists the current directory or the directory specified.

==========

3.4.4.7.4 u % Execute System Command

Format:

u%

u% command

This command allows for any system command to be executed (with the exception of **cd** in UNIX).

==========

3.4.4.7.5 u + Screendump to default printer

Format:

u+

u+ -h

u+ -Pprinter

u+ -Pprinter -h

This commmand produces hardcopy of an X-Window on the default printer or on the printer specified by the **-P** option. The **-h** option suppresses the printing of the header page.

==========

3.5 Special Considerations

3.5.1 Designing the Network

Intelligently designing the network can improve the quality and numerical stability of the simulations performed. One must always remember that the mathematical representations for devices like inductors and capacitors are only idealized approximations of the physical devices. Consequently, while we can physically stop the current in an inductor instantaneously, a simulation using an ideal inductor will fail because the voltage drop across it will become infinite. Here are a few techniques that can eliminate many of the problems of this sort:

* All inductors should have a parallel resistance to provide a path for the flyback current to flow and thereby limit the maximum voltage drop across the inductor
* Similarly, all capacitors should have a series resistance to limit the maximum current flow.
* A node connecting two switches in series should also have a resistance going to ground (or across one of the switches) to prevent a floating voltage.

3.5.2 Selection of Time Increments

Choosing an appropriate time increment is very important for ensuring an accurate simulation. If trapezoidal integration is used, a time increment that is much greater than the associated time consant for a variable will result in that variable oscillating and probably going unstable. For this case, the Euler Backward method works better since the mode is assumed to have been driven to zero for the entire time increment.

$$\frac{dx}{dt} + \frac{x}{\tau} + A = 0$$

Trapezoidal Integration

$$x - x_{old} + \left(\frac{dt}{2}\right)\left(\frac{1}{\tau}\right)(x + x_{old}) + A(dt) = 0$$

$$dt \gg \tau$$

$$x + x_{old} + 2A\tau \approx 0$$

$$x \approx -x_{old} - 2A\tau$$

Euler Backwards

$$x - x_{old} + dt\left(\frac{x}{\tau} + A\right)$$

$$dt \gg \tau$$

$$x \approx -\tau A$$

If the time step is made extremely small, the amount of computer time required to conduct the simulation becomes intolerably long with the undesired side effect of a loss of accuracy. When the time step is extremely small, round off error in the numerical calculations become a significant proportion of the corrections applied to the input variables. Over time, these errors can grow and give incorrect results.

When making the time step smaller, the CONVERGE limit must also be decreased. Otherwise, the solution for the first time increment when applied to the second time increment will result in an implicit error that remains within the CONVERGE limit. Thus, the result of the first time increment becomes the solution for the second time increment. This process is repeated for the following time increments with the net result that none of the variables deviate from their initial values.

For systems of nonlinear equations, choosing the optimal time increment is not easy to do in the general case. Usually one can get an acceptable value by trying to identify the fastest mode and using a time increment somewhat smaller than the associated time constant. Some experimentation is usually required to determine if a given choice is appropriate.

3.5.3 Using the SIMULATION File

Since virtually all of the SIMULATION Section of the input file can be edited from within SEPSIP, it is a good idea to make the entire SIMULATION section an include file. This allows one to directly save any edited parameters with the **Save SIMULATION Section (fs)** command. The next time the input file is loaded, all of the edited simulation parameters will also be loaded.

3.5.4 Using the INITIAL File

Properly using INITIAL files can greatly reduce the computational time required to conduct a simulation under certain circumstances. A typical problem may be to study the transient response of a system originally in a steady state condition that experiences some disturbance. Achieving the initial steady state condition may require a lot of simulation time due to slow "start up" time constants. To eliminate the overhead time required by the system to achieve steady state in each simulation, the INTITIAL file allows one to conduct an undisturbed simulation once, save the steady state solution in the INITIAL file, and use that INITIAL file as the starting point for all further simulation work.

3.6 Adding DEVICE Descriptions

Adding a Device description to the list of available devices requires a fair amount of effort. One or more functions must be written and compiled in the C programming language and linked with the other SEPSIP routines. Additionally, one include file (penner.h) and a device data file must be edited. Details for this procedure are contained in Appendix B.

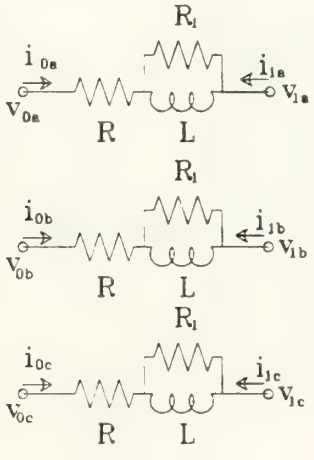
CHAPTER 4

DEVELOPMENT OF SHIPBOARD ELECTRICAL COMPONENT MODELS

4.1 Transmission Line Model

The transmission line model included in SEPSIP consists of a series combination of a resistor (**R**) and an inductor (**L**). The inductor also has another resistor ($\mathbf{R_1}$) in parallel with it to account for leakage resistance. $\mathbf{R_1}$ also helps numerically when the transmission line is attached to a switch that opens. If $\mathbf{R_1}$ were not included and the inductor current were forced to zero immediately, the voltage drop across the inductor would be infinite. $\mathbf{R_1}$ provides a path for the inductor current to flow and thereby allow a finite voltage drop. Since all physical inductors have an associated leakage resistance, the inclusion of $\mathbf{R_1}$ better reflects the actual transmission line characteristics.

Figure 4.1-1 Transmission Line



If the transmission line is excited with a sinusoidal current, the effective resistance and inductance of the transmission line is given by:

$$R_{eff} = R + R_1\left(\frac{1}{\left(\frac{R_1}{\omega L}\right)^2 + 1}\right) \qquad [1]$$

$$L_{eff} = L\left(\frac{1}{1 + \left(\frac{\omega L}{R_1}\right)^2}\right) \qquad [2]$$

The Model contains the following definitions

Parameters

R	Resistance
L	Inductance
R_1	Parallel Resistance for Inductor

Input Variables

v_{0a}	Terminal 0 Phase A Voltage
v_{0b}	Terminal 0 Phase B Voltage
v_{0c}	Terminal 0 Phase C Voltage
v_{1a}	Terminal 1 Phase A Voltage
v_{1b}	Terminal 1 Phase B Voltage
v_{1c}	Terminal 1 Phaxe C Voltage
i_{0a}	Terminal 0 Phase A Current
i_{0b}	Terminal 0 Phase B Current
i_{0c}	Terminal 0 Phase C Current
i_{1a}	Terminal 1 Phase A Current
i_{1b}	Terminal 1 Phase B Current
i_{1c}	Terminal 1 Phase C Current

State Variables

v_a	Phase A Voltage
v_b	Phase B Voltage
v_c	Phase C Voltage
i_a	Phase A Current
i_b	Phase B Current
i_c	Phase C Current

Implicit Variables

I_a	Phase A Integrator
I_b	Phase B Integrator
I_c	Phase C Integrator

Equations

$$v_a = v_{1a} - v_{0a} \quad [3]$$

$$v_b = v_{1b} - v_{0b} \quad [4]$$

$$v_c = v_{1c} - v_{0c} \quad [5]$$

$$i_a = \frac{1}{2}(i_{1a} - i_{0a}) \quad [6]$$

$$i_b = \frac{1}{2}(i_{1b} - i_{0b}) \quad [7]$$

$$i_c = \frac{1}{2}(i_{1c} - i_{0c}) \quad [8]$$

If $\mathbf{R_1}$ and **L** are both not zero, the following equations hold:

$$v_{La} = (v_a - i_a R) \quad [9]$$

$$v_{Lb} = (v_b - i_b R) \quad [10]$$

$$v_{Lc} = (v_c - i_c R) \quad [11]$$

$$i_{La} = i_a - \frac{V_{La}}{R_1} \quad [12]$$

$$i_{Lb} = i_b - \frac{V_{Lb}}{R_1} \quad [13]$$

$$i_{Lc} = i_c - \frac{V_{Lc}}{R_1} \quad [14]$$

$$I_a = i_{La} - i_{La_old} - \left(\frac{dt}{2}\right)\left(\frac{v_{La} + v_{La_old}}{L}\right) \quad [15]$$

$$I_b = i_{Lb} - i_{Lb_old} - \left(\frac{dt}{2}\right)\left(\frac{v_{Lb} + v_{Lb_old}}{L}\right) \quad [16]$$

$$I_c = i_{Lc} - i_{Lc_old} - \left(\frac{dt}{2}\right)\left(\frac{v_{Lc} + v_{Lc_old}}{L}\right) \quad [17]$$

if $\mathbf{R_1}$ or **L** are either zero, the following equations are used:

$$I_a = v_a - i_a R \quad [18]$$

$$I_b = v_b - i_b R \quad [19]$$

$$I_c = v_c - i_c R \quad [20]$$

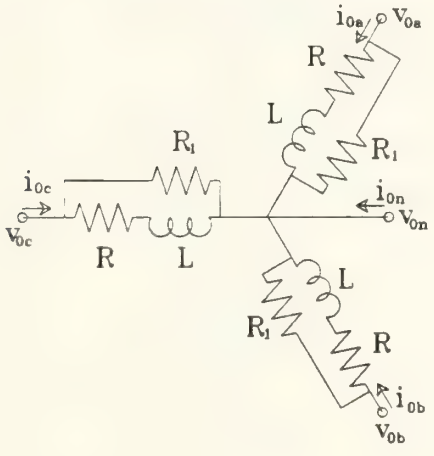
Comments

The implementation of the transmission line model is contained in the file **f_t_line_3p.c** listed in APPENDIX C.

4.2 Resistive - Reactive Load Model

The resistive - reactive load model included in SEPSIP consists of a three phase wye connected impedance. Each phase of the load consists of a resistor ($\mathbf{R_1}$) in parallel with a series combination of another resistor (**R**) and an inductance (**L**). $\mathbf{R_1}$ helps numerically when the load is attached to a switch by providing a path for the inductive current to flow and thereby prevent the voltage drop across the inductor becoming infinite.

Figure 4.2-1 Resistive Reactive Load Model



Parameters

R	Resistance (ohms)
L	Inductance (henries)
R_1	Parallel Resistance (ohms)

Input Variables

v_{0a}	Phase A Terminal Voltage
v_{0b}	Phase B Terminal Voltage
v_{0c}	Phase C Terminal Voltage
v_{0n}	Center Point Terminal Voltage
i_{0a}	Phase A Current
i_{0b}	Phase B Current
i_{0c}	Phase C Current
i_{0n}	Center Point Current

State Variables

v_a	Phase A Voltage
v_b	Phase B Voltage
v_c	Phase C Voltage

i_a Phase A Current
i_b Phase B Current
i_c Phase C Current

Implicit Variables

I_a Phase A Integrator
I_b Phase B Integrator
I_c Phase C Integrator
I_{sum} Sum of Currents

Defining Equations

$$i_{1a} = i_{0n} + i_{0b} + i_{0c} \quad [1]$$

$$i_{1b} = i_{0n} + i_{0c} + i_{0a} \quad [2]$$

$$i_{1c} = i_{0n} + i_{0a} + i_{0b} \quad [3]$$

$$v_a = v_{0a} - v_{0n} \quad [4]$$

$$v_b = v_{0b} - v_{0n} \quad [5]$$

$$v_c = v_{0c} - v_{0n} \quad [6]$$

$$i_a = \frac{i_{0a} - i_{1a}}{2} \quad [7]$$

$$i_b = \frac{i_{0b} - i_{1b}}{2} \quad [8]$$

$$i_c = \frac{i_{0c} - i_{1c}}{2} \quad [9]$$

$$I_\Sigma = i_{0a} + i_{0b} + i_{0c} + i_{0n} \quad [10]$$

If $\mathbf{L = 0}$ or $\mathbf{R_1 = 0}$ the following Equations are used:

$$R_e = \frac{R_1 R}{R_1 + R} \quad [11]$$

$$I_a = v_a - i_a R_e \quad [12]$$

$$I_b = v_b - i_b R_e \quad [13]$$

$$I_c = v_c - i_c R_e \quad [14]$$

Otherwise, the following Equations are used:

$$L\frac{d\left(i_a-\frac{v_a}{R_1}\right)}{dt}=v_a-i_aR \qquad [15]$$

$$L\frac{d\left(i_b-\frac{v_b}{R_1}\right)}{dt}=v_b-i_bR \qquad [16]$$

$$L\frac{d\left(i_c-\frac{v_c}{R_1}\right)}{dt}=v_c-i_cR \qquad [17]$$

$$I_a=i_a-i_{a_old}-\frac{v_a-v_{a_old}}{R_1}-\left(\frac{dt}{L}\right)\left(\frac{v_a+v_{a_old}}{2}-Ri_{a_old}\right) \qquad [18]$$

$$I_b=i_b-i_{b_old}-\frac{v_b-v_{b_old}}{R_1}-\left(\frac{dt}{L}\right)\left(\frac{v_b+v_{b_old}}{2}-Ri_{b_old}\right) \qquad [19]$$

$$I_c=i_c-i_{c_old}-\frac{v_c-v_{c_old}}{R_1}-\left(\frac{dt}{L}\right)\left(\frac{v_c+v_{c_old}}{2}-Ri_{c_old}\right) \qquad [20]$$

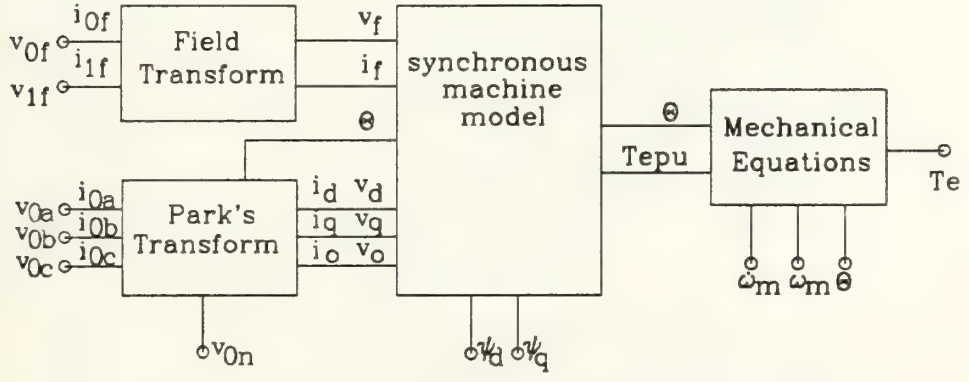
Comments

The implementation of the resistive - reactive load model is contained in the file **f_rl_wye.c** listed in APPENDIX C.

4.3 Synchronous Machine Model

Synchronous machines are used as both generators and motors onboard ships. The synchronous machine device included in SEPSIP is based on the shielding constraint model provided in [16]. This model uses the standard per unit parameters normally provided by generator manufacturers. All calculations are performed in the per unit system and employ Park's transformation to remove many of the time dependencies and thereby improve the numerical solution.[1]

Figure 4.3-1 Synchronous Machine Model



Parameters

x_d	Synchronous Reactance (PU)
x_q	Negative Sequence Reactance (PU)
x_d'	Transient Reactance (PU)
x_d''	D-axis Subtransient Reactance (PU)
x_q''	Q-axis Subtransient Reactance (PU)
x_{al}	Armature Leakage Reactance (PU)
T_{do}'	Transient Open Circuit Time Constant (sec)
T_{do}''	D-axis Subtransient OC Time Constant (sec)
T_{qo}''	Q-axis Subtransient OC Time Constant (sec)
T_{ad}	Armature Time Constant
i_{fnl}	Field Current for no load rated voltage (amps)
H	Inertia Constant (sec)
p_p	Pole Pairs

---

1 Park's Transformation expresses all of the voltages and currents in the reference frame of the rotor. The stationary reference frame is referred to as the **abc** frame while the transformed reference frame is called the **dqo** frame. Under normal operation, the **dqo** variables are not a function of the rotor angle.

ω_{ms}	Base frequency (rad/sec)
V_{db}	Base Phase Voltage (0 to peak)
P_{bs}	Base Power (watts)

Input Variables

v_{0a}	Phase A voltage
v_{0b}	Phase B voltage
v_{0c}	Phase C voltage
v_{0n}	Neutral voltage
i_{0a}	Phase A current
i_{0b}	Phase B current
i_{0c}	Phase C current
v_{0f}	Terminal 0 Field winding voltage
v_{1f}	Terminal 1 Field winding voltage
i_{0f}	Terminal 0 Field winding current
i_{1f}	Terminal 1 Field winding current
θ	Electrical rotor angle (radians)
ω_m	Mechanical frequency (rad/sec)
$\dot{\omega}_m$	Mechanical acceleration (rad/sec^2)
T_e	Torque [turbine +] (Nm)
Ψ_q	Internal Variable : q axis flux
Ψ_d	Internal Variable : d axis flux

State Variables

θ_s	Electrical Angle state (rad)
ω_{ms}	Mechanical frequency state (rad/sec)
$\dot{\omega}_{ms}$	Mechanical acceleration state (rad/sec^2)
ψ_d	D axis flux [PU]
ψ_q	Q axis flux [PU]
e_q'	Voltage behind transient reactance [PU]
e_q''	Voltage behind Q axis subtransient reactance [PU]
e_d''	Voltage behind D axis subtransient reactance [PU]
$D_{\psi d}$	Derivative of D axis flux
$D_{\psi q}$	Derivative of Q axis flux
$D_{eq'}$	Derivative of voltage behind transient reactance
$D_{eq''}$	Derivative of voltage behind Q subtransient reactance
$D_{ed''}$	Derivative of voltage behind D subtransient reactance

Implicit Variables

I_{ssum}	Sum of input phase currents
I_{fsum}	Sum of field currents
I_{vo}	neutral voltage
I_{pd}	D axis flux eqaution
I_{pq}	Q axis flux equation
$I_{eq''}$	Q axis subtransient equation
$I_{ed''}$	D axis subtransient equation
$I_{eq'}$	transient equation

I_{torq} Torq balance
I_w integrating frequency
I_{ww} integrating frequency acceleration

Equations

Calculate Base Quantities

$$I_{dB} = \frac{2}{3}\frac{P_{bs}}{V_{db}} \quad [1]$$

$$I_{fB} = I_{fnl}(x_d - x_{al}) \quad [2]$$

$$V_{fB} = \frac{P_{bs}}{I_{fB}} \quad [3]$$

$$T_{bs} = \frac{p_p P_{bs}}{\omega_{bs}} \quad [4]$$

Calculate Phase voltages

$$v_a = v_{0a} - v_{0n} \quad [5]$$

$$v_b = v_{0b} - v_{0n} \quad [6]$$

$$v_c = v_{0c} - v_{0n} \quad [7]$$

Perform Park's Transformation

$$\overline{T} = \frac{2}{3}\begin{bmatrix} \cos(\theta) & \cos\left(\theta - \frac{2\pi}{3}\right) & \cos\left(\theta + \frac{2\pi}{3}\right) \\ -\sin(\theta) & -\sin\left(\theta - \frac{2\pi}{3}\right) & -\sin\left(\theta + \frac{2\pi}{3}\right) \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \quad [8]$$

$$\begin{bmatrix} i_d \\ i_q \\ i_o \end{bmatrix} = \frac{1}{I_{dB}}\overline{T}\begin{bmatrix} i_{0a} \\ i_{0b} \\ i_{0c} \end{bmatrix} \quad [9]$$

$$\begin{bmatrix} v_d \\ v_q \\ v_c \end{bmatrix} = \frac{1}{V_{db}}\overline{T}\begin{bmatrix} v_{0a} \\ v_{0b} \\ v_{0c} \end{bmatrix} \quad [10]$$

Calculate other variables

$$i_{fd} = \frac{i_{1f} - i_{0f}}{2I_{fB}} \quad [11]$$

$$v_{fd} = \frac{v_{1f} - v_{0f}}{V_{fB}} \quad [12]$$

$$T_{aq} = \frac{x_q''}{x_d''} T_{ad} \quad [13]$$

$$x_{ad} = x_d - x_{al} \quad [14]$$

$$x_f = \frac{x_{ad}^2}{x_d - x_d'} \quad [15]$$

$$r_f = \frac{x_f}{\omega_{bs} T_{do}'} \quad [16]$$

$$x_{kd} = \frac{x_{ad}^2}{x_d - x_d''} \quad [17]$$

$$\alpha = \frac{x_d - x_d''}{x_d' - x_d''} \quad [18]$$

Calculate states

$$e_{af} = \frac{x_{ad} v_{fd}}{r_f} \quad [19]$$

$$e_d'' = x_q'' i_q - \Psi_q \quad [20]$$

$$e_q'' = -x_d'' i_d + \Psi_d \quad [21]$$

$$e_q' = \frac{x_{ad}(x_f - x_{kd}) i_{fd} + x_{kd} e_q''}{x_f} \quad [22]$$

$$\psi_d = \Psi_d \quad [23]$$

$$\psi_q = \Psi_q \quad [24]$$

Calculate Derivatives

$$D_{\psi_d} = \frac{d\psi_d}{dt} = \frac{e_q'' - \psi_d}{T_{ad}} + \omega_m p_p \psi_q + \omega_{bs} v_d \quad [25]$$

$$D_{\psi_q} = \frac{d\psi_q}{dt} = \frac{-\psi_q - e_d''}{T_{aq}} - \omega_m p_p \psi_d + \omega_{bs} v_q \quad [26]$$

$$D_{e_q''} = \frac{de_q''}{dt} = \frac{1}{T_{do}''}\left(-\frac{x_d'}{x_d''}e_q'' + e_q' + \frac{x_d' - x_d''}{x_d''}\psi_d\right) \quad [27]$$

$$D_{e_d''} = \frac{de_d''}{dt} = \frac{1}{T_{qo}''}\left(-\frac{x_q}{x_q''}e_d'' - \frac{x_q - x_q''}{x_q''}\psi_q\right) \quad [28]$$

$$D_{e_q'} = \frac{de_q'}{dt} = \frac{1}{T_{do}'}(-\alpha e_q' + (\alpha - 1)e_q'' + e_{af}) \quad [29]$$

Perform Modified Trapezoidal Integration

To determine I_{pd}, I_{pq}, $I_{eq''}$, $I_{ed''}$, and $I_{eq'}$

$$\frac{dx}{dt} = y \quad [30]$$

$$I_x = x - x_{old} - (dt)(0.6y + 0.4y_{old}) \quad [31]$$

Calculate Current Implicit Variables

$$I_{ssum} = i_o \quad [32]$$

$$I_{fsum} = \frac{i_{0f} + i_{1f}}{I_{fB}} \quad [33]$$

Calculate Mechanical Variables

$$T_{epu} = \psi_d i_q - \psi_q i_d \quad [34]$$

$$T_{acc} = \frac{2H p_p \dot{\omega}_m}{\omega_{bs}} \quad [35]$$

$$I_{torq} = T_{acc} - T_{epu} - \frac{T_e}{T_{bs}} \quad [36]$$

$$\theta_s = \theta \quad [37]$$

$$\omega_{ms} = \omega_m \quad [38]$$

$$\dot{\omega}_{ms} = \omega_{ms} \quad [39]$$

$$I_{w} = \theta_s - \theta_{s_old} - \left(\frac{dt}{2}\right)(p_p\omega_{ms} + p_p\omega_{ms_old}) \quad [40]$$

$$I_{ww} = \omega_{ms} - \omega_{ms_old} - (dt)(0.6\dot{\omega}_{ms} + 0.4\dot{\omega}_{ms_old}) \quad [41]$$

Comments

The implementation of the synchronous machine model is contained in the file **f_synch_mach.c** which is listed in APPENDIX C.

The following parameters describe a 2000 KW generator typical of those found on U.S. Navy destroyers: [10]

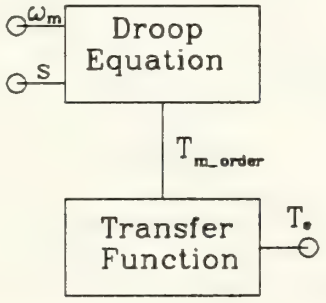
x_d	1.38	PU
x_q	0.26	PU
x_d'	0.25	PU
x_d''	0.171	PU
x_q''	0.171	PU
x_{al}	0.1	PU
T_{do}'	2.9	sec
T_{do}''	0.0	sec
T_{qo}''	0.0	sec
T_{ad}	0.09954	sec
i_{fnl}	38.5	amps
H	0.651	sec
p_p	2	
ω_{ms}	376.99	rad/sec
V_{db}	367.4235	volts
P_{bs}	2500000	watts

The subtransient time constants are set to zero becuase the author was unable to obtain their true values from unclassified sources. The time constants are typically fast and approximating them as zero does not introduce serious errors.

4.4 Speed Governor

The speed governor model included in SEPSIP is based on the mechanical speed governor on the 2000 KW synchronous steam turbine generator found on an older U.S. Navy submarine. This model is based on one developed by the author as part of a term project for an electrical engineering machinery course [10] and is a greatly reduced version of the model developed by Dalton [6].

Figure 4.4-1 Speed Governor



Parameters

ω_{nlo}	Zero Order Frequency (rad/sec)
ω_{ds}	Droop Factor Coefficient (rad/sec-inch)
ω_{dTepu}	PU Torque Coefficient (rad/sec)
T_{BS}	Base Torque (Nm)
τ_g	Time Constant (sec)

Input Variables

ω_m	Mechanical frequency (rad/sec)
T_e	Torque (Nm)
S	Droop Factor (inches)

State Variables

T_{m_order}	Ordered Torque (PU)
T_{mpu}	Actual Torque (PU)

Implicit Variables

I_g	Implicit Variable

Equations

$$T_{m_order} = \frac{1}{\omega_{wdTepu}}(\omega_m - \omega_{nlo} - \omega_{ds}s) \quad [1]$$

$$T_{mm} = T_m + B\,\omega_m \quad [2]$$

$$T_{mpu} = \frac{T_{mm}}{T_{BS}} \quad [3]$$

$$\frac{dT_{mpu}}{dt} = \frac{1}{T_g}(T_{m_order} - T_{mpu}) \quad [4]$$

$$I_g = T_g(T_{mpu} - T_{mpu_old}) - (dt)(T_{m_order_old} - 0.5T_{mpu} - 0.5T_{mpu_old}) \quad [5]$$

Comments

Equation [1] provides the steady state value for the torque corresponding to the present rotor speed. Equations [2] and [3] calculate the actual torque being delivered in the present time increment. Equation [4] describes the dynamics of the speed governor as a simple first order system. The differential equation is solved using trapezoidal integration. While this is a simple model, it does provide results consistent with the data provided in ref [6].

The Droop Factor **s** is in reality the Primary Amplifier Fulcrum Displacement. On older submarines, a stepper motor attached to a set screw is used to adjust this displacement until the desired frequency is obtained for a desired load. A normal range for **s** falls between 0 and .5 inches. The parameter values are:

$$\omega_{nlo} = 374.72$$

$$\omega_{ds} = 63.38$$

$$\omega_{dTepu} = -20.15$$

$$\tau_g = 0.328$$

The above values are for a single pole pair generator, for a two pole pair machine, the first three parameters should be halved.

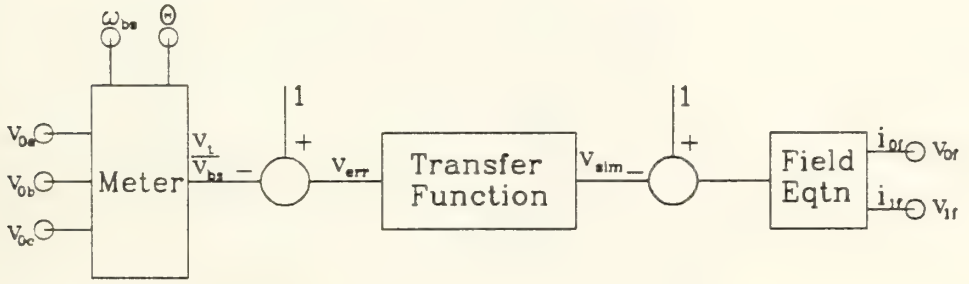
4.5 Voltage Regulator Model

The voltage regulator model included in SEPSIP is a simple first order transfer function between the terminal voltage error and field voltage of a synchronous machine. The terminal voltage is measured by subtracting the mean value of all three phases from the voltages of phases A and B. These values are fitted to two cosine voltages that are phase shifted by 120°. The derived terminal voltage is divided by the desired terminal voltage and subtracted from 1. This error voltage is subjected to a first order transfer function that produces a signal voltage that is added to 1 and multiplied by the nominal field winding voltage.

If the field voltage is driven above or below specified clipping levels, the regulator maintains the field at the clipping voltage. Typically the lower limit for the field voltage is about 0 volts while the maximum is around 1.5 to 2 times the field voltage required to maintain the terminal voltage under full load.

This model does not represent any specific voltage regulator. However, since the response of many voltage regulators is dominated by one eigenvalue, approximating the dynamics by a first order lag is not a bad assumption. The clipping action of the regulator ensures that the field voltage does not exceed reasonable bounds.

Figure 4.5-1 Voltage Regulator



Parameters

v_{fdbs}	Nominal Field winding Voltage
K	Per Unit Error Gain
T_{vr}	Voltage Regulator Time Constant (sec)
V_{fmax}	Maximum limit for field voltage
V_{fmin}	Minimum limit for field voltage (clipping)

Input Variables

v_{0a}	Phase A voltage
v_{0b}	Phase B voltage
v_{0c}	Phase C voltage
v_{0f}	Field Winding Terminal 0 voltage
v_{1f}	Field Winding Terminal 1 voltage
i_{0f}	Field Winding Terminal 0 current
i_{1f}	Field Winding Terminal 1 current
V_{bs}	Desired Voltage (neutral to line peak)
ω_{bs}	Reference Frequency (rad/sec)
v_t	Measured Voltage (internal variable)
γ	Measured Phase (internal variable)

State Variables

v_{err}	Per unit error in voltage
v_{sig}	Per unit correction to field voltage
θ	Phase (radians)
c_{clip}	Clipping State

Implicit Variables

I_1	voltmeter Phase A equation
I_2	voltmeter Phase B equation
I_{ssum}	Sum of Currents
I_n	Transfer Function Equation

Equations

Calculate the terminal voltage

$$v_n = \frac{1}{3}(v_{0a} + v_{0b} + v_{0c}) \qquad [1]$$

$$v_a = v_{0a} - v_n \qquad [2]$$

$$v_b = v_{0b} - v_n \qquad [3]$$

$$v_c = v_{0c} - v_n \qquad [4]$$

$$\gamma = \gamma_{old} + \omega_{bs}(dt) \qquad [4a]$$

$$I_1 = \frac{(v_a - v_t \cos(\theta + \gamma))}{V_{bs}} \quad [5]$$

$$I_2 = \frac{\left(v_b - v_t \cos\left(\theta + \gamma - \frac{2\pi}{3}\right)\right)}{V_{bs}} \quad [6]$$

Calculate the input and output to the transfer function

$$v_{err} = 1 - \frac{v_t}{V_{bs}} \quad [7]$$

$$v_{sig} = \frac{v_{1f} - v_{0f}}{v_{fdbs}} - 1 \quad [8]$$

If $c_{clip_old} = 0$, the regulator is not clipping:

$$T_{vr} \frac{dv_{sig}}{dt} + v_{sig} = K v_{err} \quad [9]$$

$$I_n = T_{vr}(v_{sig} - v_{sig_old}) + dt(.6(v_{sig} - K v_{err}) + .4(v_{sig_old} - K v_{err_old})) \quad [10]$$

$$v_{fd} = v_{1f} - v_{0f} \quad [11]$$

otherwise the regulator is clipping

$$\text{if } c_{clip_old} = -1 \text{ then } I_n = \frac{V_{fmin} - (v_{1f} - v_{0f})}{V_{bs}} \quad [12]$$

$$\text{if } c_{clip_old} = 1 \text{ then } I_n = \frac{V_{fmax} - (v_{1f} - v_{0f})}{V_{bs}} \quad [13]$$

$$v_{sig} = \frac{T_{vr} V_{sig_old} - dt[.4(V_{sig_old} - K V_{err_old}) - .6 K V_{err}]}{T_{vr} + .6dt} \quad [14]$$

$$v_f = (v_{sig} + 1) v_{fdbs} \quad [15]$$

The sum of the field currents should be zero

$$I_{ssum} = i_{0f} + i_{1f} \quad [16]$$

See if should clip during the next time increment

$$\text{if } v_f \geq v_{fmax} \text{ then } c_{clip} = 1 \quad [17]$$

$$\text{else if } v_f \leq v_{fmin} \text{ then } c_{clip} = -1 \quad [18]$$

$$\text{else } c_{clip} = 0 \quad [19]$$

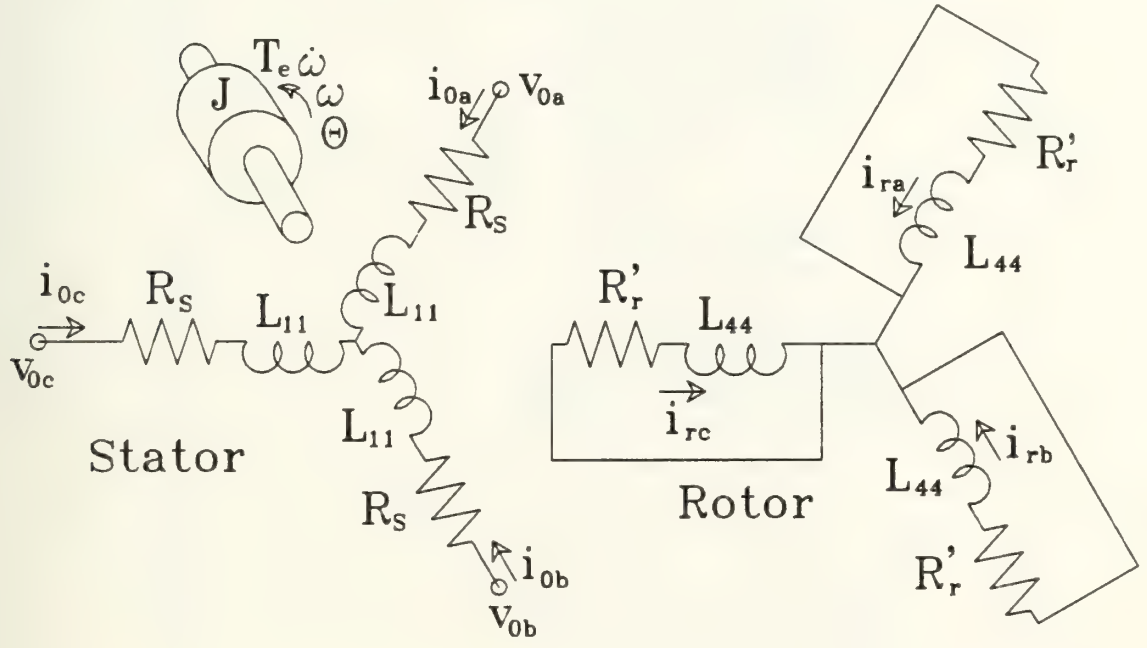
Comments

The implementation of the voltage regulator model is contained in the file **f_volt_reg.c** listed in APPENDIX C.

4.6 Induction Motor Model

The Induction Motor Device included in SEPSIP is based on a model of a "squirrel cage" motor presented by Krause [17]. This model assumes the stator consists of three windings and that the rotor can be represented by three additional windings.

Figure 4.6-1 Induction Motor



Mutual Inductances Not Shown

Parameters

R_s	Stator Resistance (ohms)
X_{ls}	Stator Inductance (ohms)
X_{lr}'	Rotor Inductance (reflected to Stator) (ohms)
R_r'	Rotor Resistance (reflected to Stator) (ohms)
J	Moment of Inertia (Kg-m^2)
ω_{bs}	Base Frequency (rad/sec)
p_p	Pole Pairs
B	Windage Torque Factor (Nm-sec)

Input Variables

v_{0a}	Phase A voltage
v_{0b}	Phase B voltage
v_{0c}	Phase C voltage
i_{0a}	Phase A current
i_{0b}	Phase B current
i_{0c}	Phase C current
θ	Electrical Rotor Angle (radians)
ω	Mechanical Frequency (rad/sec)
$\dot{\omega}$	Mechanical Acceleration (rad/sec^2)
v_{0n}	Neutral voltage
i_{ra}	Phase A rotor current
i_{rb}	Phase B rotor current
i_{rc}	Phase C rotor current

State Variables

λ_{sa}	Phase A stator flux
λ_{sb}	Phase B stator flux
λ_{sc}	Phase C stator flux
D_{sa}	Phase A stator flux derivative
D_{sb}	Phase B stator flux derivative
D_{sc}	Phase C stator flux derivative
λ_{ra}'	Phase A rotor flux
λ_{rb}'	Phase B rotor flux
λ_{rc}'	Phase C rotor flux
D_{ra}	Phase A rotor flux derivative
D_{rb}	Phase B rotor flux derivative
D_{rc}	Phase C rotor flux derivative
θ_s	Electrical Rotor Angle (rad)
ω_s	Mechanical Speed (rad/sec)
$\dot{\omega}_s$	Mechanical Acceleration (rad/sec)

Implicit Variables

I_{la}	Phase A stator flux equation
I_{lb}	Phase B stator flux equation
I_{lc}	Phase C stator flux equation
I_{lra}	Phase A rotor flux equation
I_{lrb}	Phase B rotor flux equation
I_{lrc}	Phase C rotor flux equation
I_s	Sum of Stator Currents
W	Frequency Integration
W_D	Acceleration Integration
I_T	Torque equation

Equations

Calculate Inductances

$$L_{ls} = \frac{X_{ls}}{\omega_{bs}} \quad [1]$$

$$L_{lr}' = \frac{X_{lr}'}{\omega_{bs}} \quad [2]$$

$$M = \frac{X_M}{\omega_{bs}} \quad [3]$$

$$L_{ms} = \frac{2}{3} M \quad [4]$$

$$L_{11} = L_{ls} + L_{ms} \quad [5]$$

$$L_{44} = L_{lr}' + L_{ms} \quad [6]$$

$$L_{21} = -\frac{L_{ms}}{2} \quad [7]$$

$$L_{41} = L_{ms} \cos(\theta) \quad [8]$$

$$L_{51} = L_{ms} \cos\left(\theta + \frac{2\pi}{3}\right) \quad [9]$$

$$L_{61} = L_{ms} \cos\left(\theta - \frac{2\pi}{3}\right) \quad [10]$$

Calculate Fluxes

$$\lambda_{sa} = L_{11} i_{0a} + L_{21} i_{0b} + L_{21} i_{0c} + L_{41} i_{ra} + L_{51} i_{rb} + L_{61} i_{rc} \quad [11]$$

$$\lambda_{sb} = L_{21} i_{0a} + L_{11} i_{0b} + L_{21} i_{0c} + L_{61} i_{ra} + L_{41} i_{rb} + L_{51} i_{rc} \quad [12]$$

$$\lambda_{sc} = L_{21} i_{0a} + L_{21} i_{0b} + L_{11} i_{0c} + L_{51} i_{ra} + L_{61} i_{rb} + L_{41} i_{rc} \quad [13]$$

$$\lambda_{ra}' = L_{41} i_{0a} + L_{61} i_{0b} + L_{51} i_{0c} + + L_{44} i_{ra} + L_{21} i_{rb} + L_{21} i_{rc} \quad [14]$$

$$\lambda_{rb}' = L_{51} i_{0a} + L_{41} i_{0b} + L_{61} i_{0c} + + L_{21} i_{ra} + L_{44} i_{rb} + L_{21} i_{rc} \quad [15]$$

$$\lambda_{rc}' = L_{61} i_{0a} + L_{51} i_{0b} + L_{41} i_{0c} + + L_{21} i_{ra} + L_{21} i_{rb} + L_{44} i_{rc} \quad [16]$$

Calculate Flux Derivatives

$$D_{sa} = v_{0a} - v_{0n} - R_s i_{0a} \quad [17]$$

$$D_{sb} = v_{0b} - v_{0n} - R_s i_{0b} \quad [18]$$

$$D_{sc} = v_{0c} - v_{0n} - R_s i_{0c} \quad [19]$$

$$D_{ra} = -R_r' i_{ra} \quad [20]$$

$$D_{rb} = -R_r' i_{rb} \quad [21]$$

$$D_{rc} = -R_r' i_{rc} \quad [22]$$

Perform Trapezoidal Integration

$$I_{la} = \lambda_{sa} - \lambda_{sa_old} - \left(\frac{dt}{2}\right)(D_{sa} + D_{sa_old}) \quad [23]$$

$$I_{lb} = \lambda_{sb} - \lambda_{sb_old} - \left(\frac{dt}{2}\right)(D_{sb} + D_{sb_old}) \quad [24]$$

$$I_{lc} = \lambda_{sc} - \lambda_{sc_old} - \left(\frac{dt}{2}\right)(D_{sc} + D_{sc_old}) \quad [25]$$

$$I_{lra} = \lambda_{ra}' - \lambda_{ra_old}' - \left(\frac{dt}{2}\right)(D_{ra} + D_{ra_old}) \quad [26]$$

$$I_{lrb} = \lambda_{rb}' - \lambda_{rb_old}' - \left(\frac{dt}{2}\right)(D_{rb} + D_{rb_old}) \quad [27]$$

$$I_{lrc} = \lambda_{rc}' - \lambda_{rc_old}' - \left(\frac{dt}{2}\right)(D_{rc} + D_{rc_old}) \quad [28]$$

$$I_s = i_{0a} + i_{0b} + i_{0c} \quad [29]$$

Calculate Torque and Mechanical variables

$$T_e = -p_p L_{ms}\{\left(i_{0a}\left(i_{ra} - \frac{i_{rb}}{2} - \frac{i_{rc}}{2}\right) + i_{0b}\left(i_{rb} - \frac{i_{rc}}{2} - \frac{i_{ra}}{2}\right) + i_{0c}\left(i_{rc} - \frac{i_{ra}}{2} - \frac{i_{rb}}{2}\right)\right)\sin(\theta) +$$

$$\sqrt{\frac{3}{2}}(i_{0a}(i_{rb} - i_{rc}) + i_{0b}(i_{rc} - i_{ra}) + i_{0c}(i_{ra} - i_{rb}))\cos(\theta)\} \quad [30]$$

$$I_T = T_e - J\dot{\omega} - T_{mech} - B\omega \quad [31]$$

$$\theta_s = \theta \quad [32]$$

$$\omega_s = \omega \quad [33]$$

$$\dot{\omega}_s = \dot{\omega} \quad [34]$$

$$W = \theta_s - \theta_{s_old} - \left(\frac{dt}{2}\right) p_p(\omega_s + \omega_{s_old}) \quad [35]$$

$$W_D = \omega_s - \omega_{s_old} - \left(\frac{dt}{2}\right)(\dot{\omega}_s + \dot{\omega}_{s_old}) \quad [36]$$

Comments

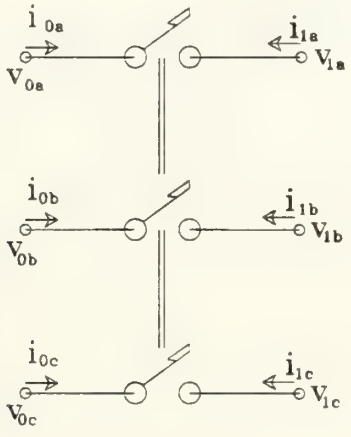
The implementation of the induction motor model is contained in the file **f_ind_motor.c** listed in APPENDIX C.

4.7 Three Phase Switch Model

The Three Phase Switch model allows the user to control switches for all three phases with one external input variable. When the external input variable commands the switches to close, all three phase switches close at once. When commanded to open however, the individual phase switches remain closed until a zero crossing occurs.

Since a switch changes the configuration of the network, one must ensure that both configurations are defined properly and have a voltage reference. In particular, switches should not be connected in series without providing some means (such as a resistor) to define the voltage of the node connecting the two switches.

Figure 4.7-1 Three Phase Switch



Input Variables

v_{0a}	Terminal 0 Phase A Voltage
v_{0b}	Terminal 0 Phase B Voltage
v_{0c}	Terminal 0 Phase C Voltage
v_{1a}	Terminal 1 Phase A Voltage
v_{1b}	Terminal 1 Phase B Voltage
v_{1c}	Terminal 1 Phase C Voltage
i_{0a}	Terminal 0 Phase A Current
i_{0b}	Terminal 0 Phase B Current
i_{0c}	Terminal 0 Phase C Current
i_{1a}	Terminal 1 Phase A Current
i_{1b}	Terminal 1 Phase B Current
i_{1c}	Terminal 1 Phase C Current

State Variables

s_a	Phase A state
s_b	Phase B state
s_c	Phase C state
i_a	Phase A current
i_b	Phase B current
i_c	Phase C current

Implicit

I_{sa}	Phase A switch
I_{sb}	Phase B switch
I_{sc}	Phase C switch
I_{ia}	Phase A current sum
I_{ib}	Phase B current sum
I_{ic}	Phase C current sum

External Input

S	Switch Condition 0 = on 1 = off

Equations

$$v_a = v_{1a} - v_{0a} \quad [1]$$

$$v_b = v_{1b} - v_{0b} \quad [2]$$

$$v_c = v_{1c} - v_{0c} \quad [3]$$

$$i_a = \frac{1}{2}(i_{1a} - i_{0a}) \quad [4]$$

$$i_b = \frac{1}{2}(i_{1b} - i_{0b}) \quad [5]$$

$$i_c = \frac{1}{2}(i_{1c} - i_{0c}) \quad [6]$$

If the Switch is commanded closed:

$$s_a = s_b = s_c = 1 \quad [7]$$

If the Switch is commanded opened:

$$s_a = s_{a_old} \quad [8]$$

$$s_b = s_{b_old} \quad [9]$$

$$s_c = s_{c_old} \quad [10]$$

Calculate the implicit variables:

$$\text{if } s_a = 1 \text{ then } I_{sa} = v_a \text{ else } I_{sa} = i_a \quad [11]$$

$$\text{if } s_b = 1 \text{ then } I_{sb} = v_b \text{ else } I_{sb} = I_b \quad [12]$$

$$\text{if } s_c = 1 \text{ then } I_{sc} = v_c \text{ else } I_{sc} = I_c \quad [13]$$

$$I_{ia} = i_{0a} + i_{1a} \quad [14]$$

$$I_{ib} = i_{0b} + i_{1b} \quad [15]$$

$$I_{ic} = i_{0c} + i_{1c} \quad [16]$$

If the Switch is commanded opened, Check for Zero Crossing:

$$\text{if } i_a i_{a_old} \le 0 \text{ then } s_a = 0 \text{ else } s_a = 1 \quad [17]$$

$$\text{if } i_b i_{b_old} \le 0 \text{ then } s_b = 0 \text{ else } s_b = 1 \quad [18]$$

$$\text{if } i_c i_{c_old} \le 0 \text{ then } s_c = 0 \text{ else } s_c = 1 \quad [19]$$

Comments

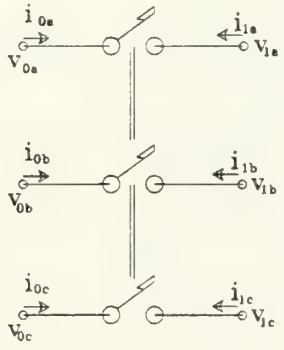
The implementation of the three phase switch is contained in the file **f_switch_3p.c** contained in APPENDIX C.

Note that the switch does not open until a zero crossing has occurred. This may lead to problems when the derivative of the current is very large and the current overshoots zero by a large amount.

4.8 Circuit Breaker Model

The circuit breaker model included in SEPSIP simulates a three phase circuit breaker that is tripped either by an overcurrent or 'manually' by the operator. The square of the magnitude of the current is determined through a forgetting factor **f** which is multiplied by the square of the old value of the current magnitude and added to **(1 - f)** times the square of the present current value. The magnitude of the current is compared to a specified limit, if the limit has been exceeded for a certain amount of time, the breaker is commanded to open. The breaker can also be commanded to open by an external input. Once commanded to open, the switch for each phase remains closed until a zero crossing occurs.

Figure 4.8-1 Circuit Breaker



Each phase of the circuit breaker is modelled as a state machine having seven states defined by:

Figure 4.8-2 Circuit Breaker States			
State	Switch	External Cmd	Current Trip
0	open	open	open
1	closed	open	open
2	open	open	closed
3	closed	open	closed
4	open	closed	open
5	closed	closed	open
7	closed	closed	closed

Parameters

f	RMS Forgetting Factor
I_{trip}	Current Trip value (amps rms)
t_{trip}	Current Trip Minimum Time (sec)

Input Variables

v_{0a}	Terminal 0 Phase A Voltage
v_{0b}	Terminal 0 Phase B Voltage
v_{0c}	Terminal 0 Phase C Voltage
v_{1a}	Terminal 1 Phase A Voltage
v_{1b}	Terminal 1 Phase B Voltage
v_{1c}	Terminal 1 Phase C Voltage
i_{0a}	Terminal 0 Phase A Current
i_{0b}	Terminal 0 Phase B Current
i_{0c}	Terminal 0 Phase C Current
i_{1a}	Terminal 1 Phase A Current
i_{1b}	Terminal 1 Phase B Current
i_{1c}	Terminal 1 Phase C Current

States

s_a	Phase A switch state
s_b	Phase B switch state
s_c	Phase C switch state
i_a	Phase A current
i_b	Phase B current
i_c	Phase C current
i_{aave}	Phase A rms current
i_{bave}	Phase B rms current
i_{cave}	Phase C rms current
t_{ia}	Phase A overcurrent time
t_{ib}	Phase B overcurrent time
t_{ic}	Phase C overcurrent time

External Input Variables

Switch	External Switch

Implicit Variables

I_{sa}	Phase A switch equation
I_{sb}	Phase B switch equation
I_{sc}	Phase C switch equation
I_{ia}	Phase A current sum
I_{ib}	Phase B current sum
I_{ic}	Phase C current sum

Equations

$$v_a = v_{1a} - v_{0a} \quad [1]$$

$$v_b = v_{1b} - v_{0b} \quad [2]$$

$$v_c = v_{1c} - v_{0c} \quad [3]$$

$$i_a = \frac{1}{2}(i_{1a} - i_{0a}) \quad [4]$$

$$i_b = \frac{1}{2}(i_{1b} - i_{0b}) \quad [5]$$

$$i_c = \frac{1}{2}(i_{1c} - i_{0c}) \quad [6]$$

Perform the following State Transformation:

Figure 4.8-3 Breaker Transform Table 1		
Old State	Switch on	Switch off
0	7	0
1	7	1
2	7	2
3	7	3
4	4	0
5	5	1
7	7	3

If $\mathbf{t}_{ia_old}$, $\mathbf{t}_{ib_old}$, or $\mathbf{t}_{ic_old}$ is greater than $\mathbf{t}_{trip}$

use the following transition table

Figure 4.8-4 Breaker Transform Table 2	
Old State	New State
0	0
1	1
2	0
3	1
4	4
5	5
7	5

The Implicit variables are defined by

if s_a is odd then $I_{sa} = v_a$ else $I_{sa} = i_a$ [7]

if s_b is odd then $I_{sb} = v_b$ else $I_{sb} = i_b$ [8]

if s_c is odd then $I_{sc} = v_c$ else $I_{sc} = i_c$ [9]

$$I_{ia} = i_{0a} + i_{1a} \quad [10]$$

$$I_{ib} = i_{0b} + i_{1b} \quad [11]$$

$$I_{ic} = i_{0c} + i_{1c} \quad [12]$$

Look For Zero Crossing

If the product of a phase current's present and old values is less than or equal to zero, perform this transformation:

Figure 4.8-5 Breaker Transform Table 3	
Old State	New State
0	0
1	0
2	2
3	2
4	4
5	4
7	7

Calculate RMS Currents

$$i_{aave} = \sqrt{(1-f)i_a^2 + f i_{aave_old}^2} \quad [13]$$

$$i_{bave} = \sqrt{(1-f)i_b^2 + f i_{bave_old}^2} \quad [14]$$

$$i_{cave} = \sqrt{(1-f)i_c^2 + f i_{cave_old}^3} \quad [15]$$

Update Overcurrent Time counters

$$\text{if } i_{aave} \geq i_{trip} \text{ then } t_{ia} = t_{ia_old} + dt \text{ else } t_{ia} = 0 \quad [16]$$

$$\text{if } i_{bave} \geq i_{trip} \text{ then } t_{ib} = t_{ib_old} + dt \text{ else } t_{ib} = 0 \quad [17]$$

$$\text{if } i_{cave} \geq i_{trip} \text{ then } t_{ic} = t_{ic_old} + dt \text{ else } t_{ic} = 0 \quad [18]$$

Comments

The implementation of the three phase circuit breaker model is contained in the file **f_breaker_3p.c** listed in APPENDIX C.

CHAPTER 5

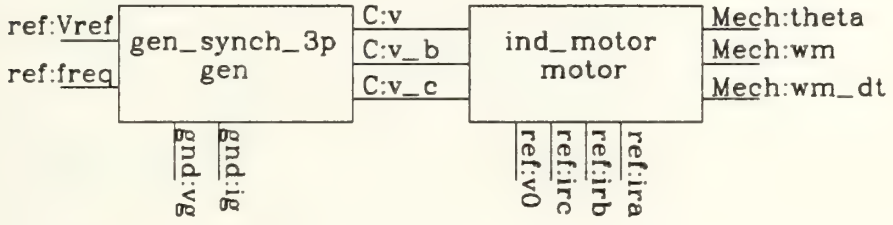
SIMULATION RESULTS

A description of SEPSIP would be incomplete without several examples of simulations conducted with the program. This chapter contains the input files and results of seven simulations that demonstrate the features and capabilities of SEPSIP. The first two simulations show the start up transients for two different induction motors. Because the simulations are very similar, only slight modifications to the input files of the first simulation were required to generate the second simulation. The remaining simulations involve the response of one or two synchronous generators to changing loads. The third simulation loads a single synchronous generator with a .4 Per Unit load for a 4 second period and then removes the load. The response of the voltage regulator and speed governor to the application and removal of the load are clearly shown. The next two simulations use identical network topologies to study a generator's response to two and three phase shorts. The input files for these simulations were modified from the third simulation in a very short time. The final two simulations demonstrate the ability of SEPSIP to simulate the dynamics of multiple generator systems.

5.1 50 HP Induction Motor Start Up

This simulation shows the startup transients for a 50 HP Induction motor. The network consists of a simple three phase generator directly attached to the induction motor. Each phase of the generator consists of a sinusoidal source in series with a parallel combination of a resistor and an inductor. Figure 5.1-1 shows the structure of the network.

Figure 5.1-1 50 HP Induction Motor



The parameters for the induction motor are from Krause [17]. The Element description include file for this simulation is shown in figure 5.1-2

Figure 5.1-2 t50.elm : Element Description File

```
! t50.elm
! Norbert H. Doerry
!
! This file defines the elements for a simple 3 phase generator attached
! to an induction motor.
!
gen_synch_3p gen
  phase_a 0.0
  L 0.0001
  R 1000
  end
!
! The following are the characteristics of a 50 HP motor
! described on page 190 of Krause's ANALYSIS OF ELECTRIC MACHINERY
!
ind_motor motor
  Rs            .0807
  Xls           .302
  XM          13.08
  Xlr_prime     .302
  Rr_prime      .228
  J            1.662
  wbs       376.99
  pp            2
  B             0.0
  end
!
```

Note that the series induction for the generator is very small. This was done to simulate a voltage bus that was almost infinite in nature. (The voltage drop due to the synchronous reactance is negligible).

The effect of setting **B** to zero for the induction motor is to ignore the windage losses. Since Krause also ignored windage losses in his analysis, the results from this simulation can be directly compared.

Figure 5.1-3 t50.net : Network Description File

```
! t50.net
! Norbert H. Doerry
!
NETWORK
!
NODE gnd
  rv:ig = 0 = gen:i0n
  rv:vg = 0 = gen:v0n
  end
NODE C
  v3:v   = gen:v0a = motor:v0a
  i3:i   = gen:i0a = motor:i0a
  end
NODE Mech
  v:theta = motor:theta
  v:wm    = motor:wm
  v:wm_dt = motor:wm_dt
  end
NODE ref
  v:ira   = motor:ira
  v:irb   = motor:irb
  v:irc   = motor:irc
  v:v0    = motor:v0n
 rv:Vref  = gen:Vmag
 rv:freq  = 60.0 = gen:freq
 end
!
```

Figure 5.1-4 t50.init : Initialization File

```
! t50.init
! Norbert H. Doerry
!
INITIALIZE
  END
NODE VOLTAGE INITIALIZATION
  C:v          375.6
  C:v_b       -187.8
  C:v_c       -187.8
  END
EXTERNAL INPUTS INITIALIZATION
  motor:Tmech    0.0
  END
```

Figure 5.1-5 t50.sim : Simulation File

```
! t50.sim
! Norbert H. Doerry
!
SIMULATION
!
DISPLAY
  motor:RPM
  motor:HP
  motor:Te
  gen  :Ia
  gen  :Ib
  gen  :Ic
  gen  :Va
  ref  :ira
  ref  :irb
  ref  :irc
  END
!
  TIME_STEP  0.00025
  TMIN       0
  TMAX       1.0
  PRINT_STEP 0.0010
  DELTA      0.01
  DELTA_MIN  0.01
  CONVERGE   1e-10
  MAX_ITERATION 50
  REFERENCE
    I:  C:i
    V:ref:Vref   375.6
    END
!
EXTERNAL INPUTS
  END
!
```

The above files are synthesized into a single input file with the following format:

Figure 5.1-6 t50.all : Input File

```
! t50.all
! Norbert H. Doerry
!
include t50.elm
include t50.net
include t50.init
include t50.sim
!
```

The above format for the input file was used for all the simulations conducted for this thesis. In the following sections, the include files will be referred to as separate entities even though they only have meaning when organized in the manner shown in figure 5.1-6. SEPSIP does not require the organization of the input file in this manner, but this convention greatly eases the task of creating and running simulations.

The results of the simulation are shown in figures 5.1-7 through 5.1-10. Note that for this motor, the machine reaches its steady state speed after only .6 seconds. As typically seen

in many induction machines of this size, the transient currents during the startup are considerably greater than the final no load current. The rotor current also shows the expected characteristic decreasing frequency as the rotor approaches synchronous speed. These results are all identical to the figures shown in reference [17].

Fig 5.1-7 RPM vs Time

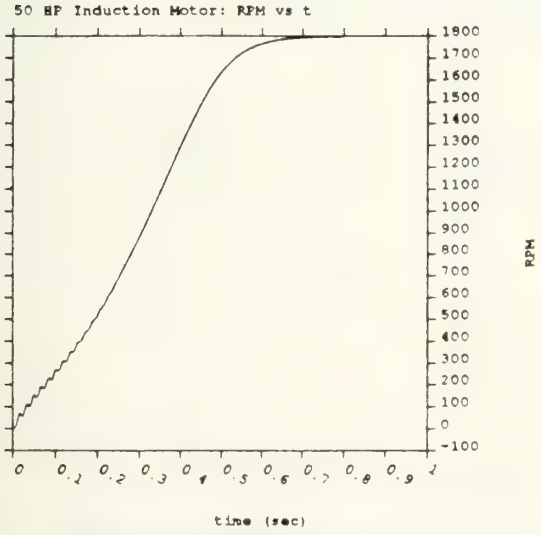


Fig 5.1-8 Te vs RPM

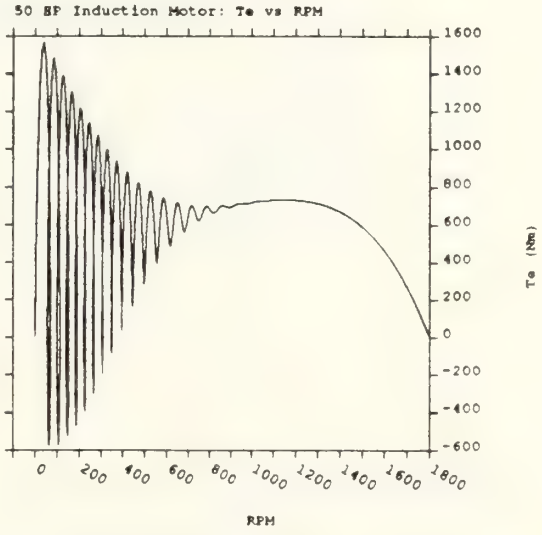


Fig 5.1-9 Stator Current vs Time

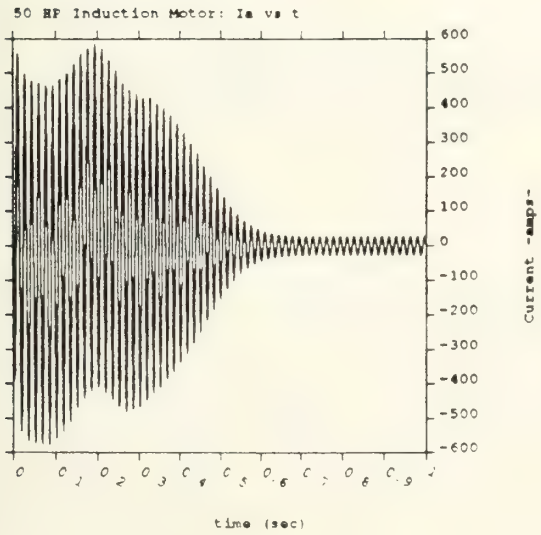
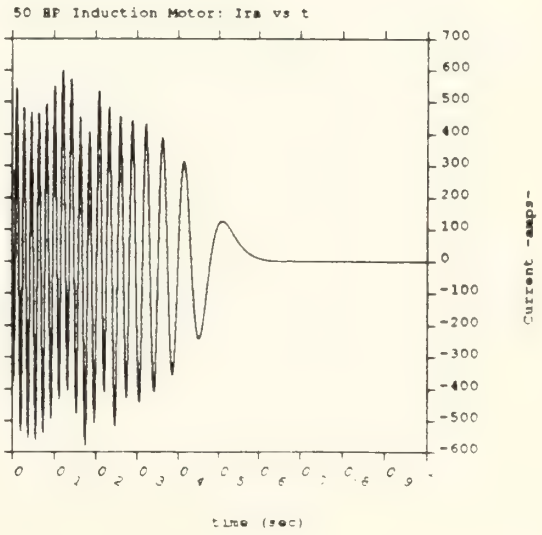


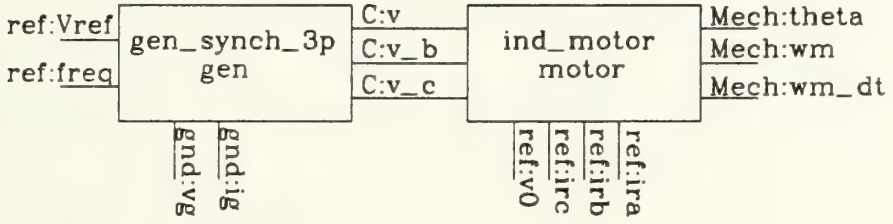
Fig 5.1-10 Rotor Current vs Time



5.2 500 HP Induction Motor Start Up

This simulation shows the startup transients for a 500 HP Induction motor. The network consists of a simple three phase generator directly attached to the induction motor and is identical to the network used for the 50 HP Induction motor in the previous section. Figure 5.2-1 shows the structure of the network.

Figure 5.2-1 500 HP Induction Motor



The parameters for the induction motor are from Krause [17]. The Element description include file for this simulation is shown in figure 5.2-2

Figure 5.2-2 t500.elm : Element Description File

```
! t500.elm
! Norbert H. Doerry
!
! This file defines the elements for a simple 3 phase generator attached
! to an induction motor.
!
gen_synch_3p gen
  phase_a 0.0
  L 0.0001
  R 1000
  end
!
! The following are the characteristics of a 500 HP motor
! described on page 190 of Krause's ANALYSIS OF ELECTRIC MACHINERY
!
ind_motor motor
  Rs            .262
  Xls          1.206
  XM          54.02
  Xlr_prime    1.206
  Rr_prime      .187
  J           11.06
  wbs        376.99
  pp             2
  B            0.0
  end
!
```

Note that the series induction for the generator is very small. This was done for the same reasons discussed in the 50 HP induction motor example.

Setting **B** to zero results once again with the induction motor dynamics ignoring the windage losses. Since Krause also ignored windage losses in his analysis, the results from this simulation can be directly compared.

Figure 5.2-3 t500.net : Network Description File

```
! t500.net
! Norbert H. Doerry
!
NETWORK
!
NODE gnd
  rv:ig = 0 = gen:i0n
  rv:vg = 0 = gen:v0n
  end
NODE C
  v3:v   = gen:v0a = motor:v0a
  i3:i   = gen:i0a = motor:i0a
  end
NODE Mech
  v:theta = motor:theta
  v:wm    = motor:wm
  v:wm_dt = motor:wm_dt
  end
NODE ref
  v:ira   = motor:ira
  v:irb   = motor:irb
  v:irc   = motor:irc
  v:v0    = motor:v0n
 rv:Vref  = gen:Vmag
 rv:freq  = 60.0 = gen:freq
 end
!
```

Figure 5.2-4 t500.init : Initialization File

```
! t500.init
! Norbert H. Doerry
!
INITIALIZE
  END
NODE VOLTAGE INITIALIZATION
  C:v         1877.9
  C:v_b       -938.95
  C:v_c       -938.95
  END
EXTERNAL INPUTS INITIALIZATION
  motor:Tmech    0.0
  END
```

Figure 5.2-5 t500.sim : Simulation File

```
! t500.sim
! Norbert H. Doerry
!
SIMULATION
!
DISPLAY
  motor:RPM
  motor:HP
  motor:Te
  gen  :Ia
  gen  :Ib
  gen  :Ic
  gen  :Va
  ref  :ira
  ref  :irb
  ref  :irc
  END
!
  TIME_STEP  0.00025
  TMIN       0
  TMAX       2.0
  PRINT_STEP 0.0010
  DELTA      0.01
  DELTA_MIN  0.01
  CONVERGE   1e-10
  MAX_ITERATION 50
  REFERENCE
    I:  C:i
    V:ref:Vref   1877.9
    END
!
EXTERNAL INPUTS
  END
!
```

The results of the simulation are shown in figures 5.2-6 through 5.2-9. Note that for this motor, the machine reaches its steady state speed after about 1.4 seconds. As typically seen in many induction machines of this size, the transient currents during the startup are considerably larger than the final no load current. The large startup currents indicate that a motor controller should be used during start up. The rotor current also shows the expected characteristic decreasing frequency as the rotor approaches synchronous speed. Another typical characteristic of large induction motors is the speed overshoot shown in figures 5.2-6 and 5.2-7. Note that the 50 HP machine did not have an overshoot. These results are all identical to the figures shown in reference [17].

Fig 5.2-6 RPM vs Time

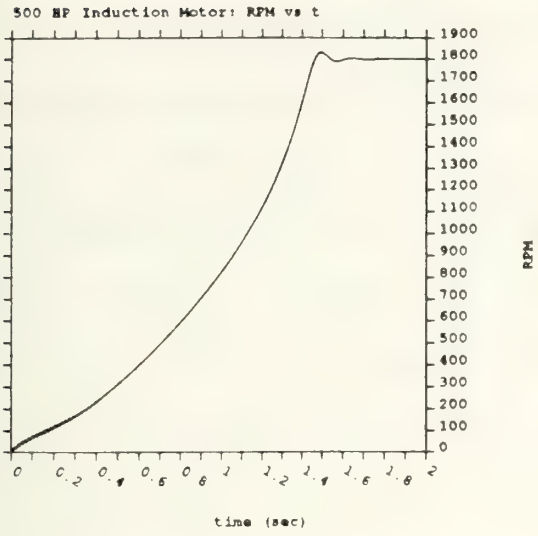


Fig 5.2-7 Te vs RPM

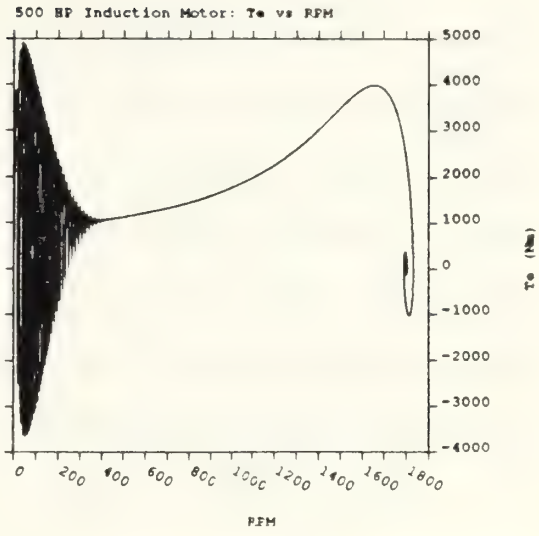


Fig 5.2-8 Stator Current vs Time

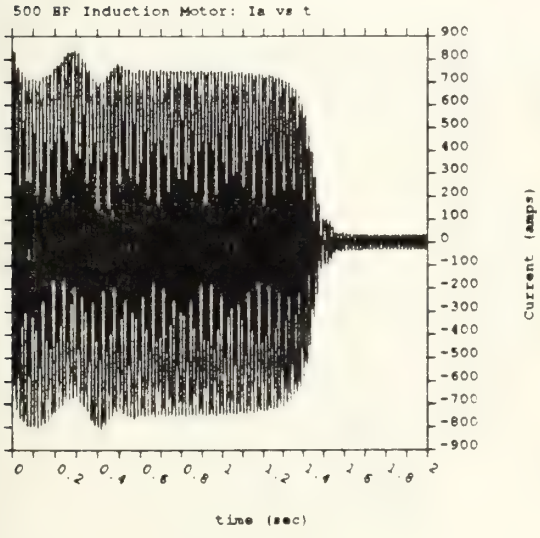
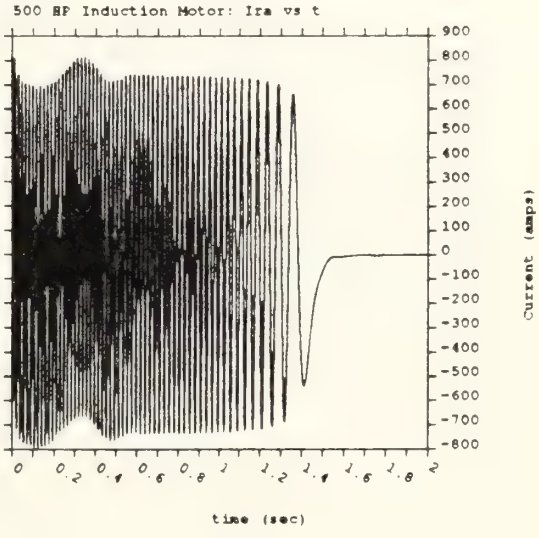


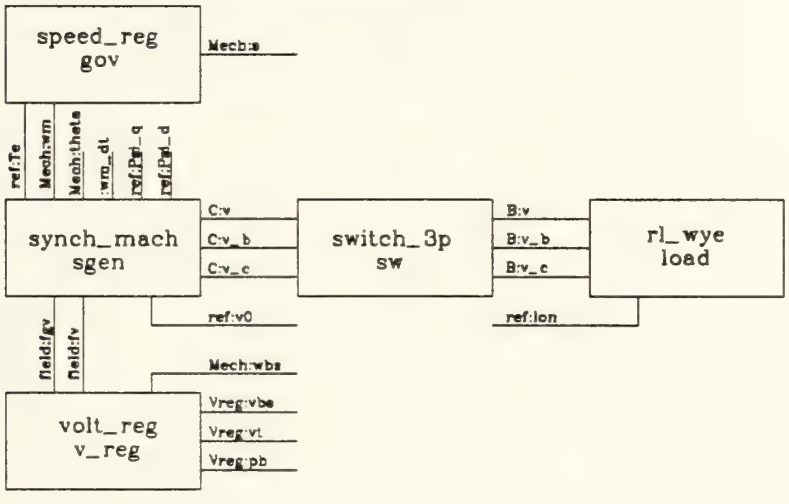
Fig 5.2-9 Rotor Current vs Time



5.3 Synchronous Generator: Switched Load

This simulation demonstrates the transients associated with loading an initially unloaded synchronous generator to forty percent of its rated load. The synchronous generator is modelled on a 2000 KW unit found on recent construction guided missile destroyers (DDG). Speed regulation is performed by a droop governor based on the type used on an older submarine.[1] The voltage regulator is modelled as a first order lag transfer function with output clipping. While most voltage regulators have more complicated transfer functions, this model provides reasonable results under normal operating conditions. The load is a wye connected impedance with a .999 power factor. Figure 5.3-1 shows the structure of the network:

Figure 5.3-1 Synchronous Generator: Switched Load



1 A droop governor has a torque speed characteristic that has a negative sloop. This implies that as the load on the generator increases, the frequency decreases. Another type of speed regulator, the isosynchronous governor, maintains a constant speed regardless of the load. In isolated operation, most modern generators operate in the isosynchronous mode. When two or more generators are paralleled however, the droop mode provides a inherently stable method for sharing the load. Paralleled isosynchronous generators require special circuitry to ensure one generator does not take all of the load.

The parameters for the generator were obtained from reference [10]. The voltage regulator dynamics were extracted from data provided by references [2] and [18]. Speed governor dynamics are from references [6] and [10]. The parameters are all listed in figure 5.3-2

Figure 5.3-2 v.elm : Element Description File

```
!v.elm
!Norbert H. Doerry
!
synch_mach sgen
  xd       1.38
  xq       0.26
  xd_p     0.25
  xd_pp    0.171
  xq_pp    0.171
  xal      0.1     ! This is an approximation
  Tdo_p    2.9
  Tdo_pp   0.0     ! No data for subtransient T.C.
  Tqo_pp   0.0
  Tad      0.09954
  Ifnl    38.5     ! Field Current for no load rated terminal voltage
  H        0.651
  pp       2
  wbs    376.99
  Vdb    367.4235  ! 450 volts rms line to line
  Pbs    2.5e6     ! 2000 KW at .8 Power Factor
  end
!
rl_wye load
  R 0.2
  L 25e-6
  R1 1000
  end
!
switch_3p sw
  end
!
speed_reg gov
  wnlo      187.36
  wds        31.69
  wdTepu    -10.07
  TBS       13262.6
  Tg          0.3275
  B           0        ! damping is ignored
  end
!
volt_reg v_reg
  Vfdbs      52.56  ! Field Voltage for rated noload terminal voltage
  K          20.0   ! Transfer function gain
  Tvr        0.15   ! Principle time constant
  Vfmin      0      ! Minimum field voltage
  Vfmax      120.0  ! Maximum field voltage (clipping)
  end
```

Figure 5.3-3 v.net : Network Description File

```
!v.net
!Norbert H. Doerry
!
NODE C
  v3:v = sgen:v0a = sw:v0a = v_reg:v0a
  i3:i = sgen:i0a = sw:i0a
  end
NODE B
```

```
  v3:v = sw:v1a = load:v0a
  i3:i = sw:i1a = load:i0a
  end
NODE Mech
   v:theta = sgen:theta
  rv:wbs   = 377     = v_reg:wbs
   v:wm    = sgen:wm = gov:wm
   v:wm_dt = sgen:wm_dt
  rv:s     = 0.04    = gov:s
  end
NODE ref
   v:Te    = sgen:Te = gov:Tm
   v:Psi_q = sgen:Psi_q
   v:Psi_d = sgen:Psi_d
  rv:v0    = 0.0 = sgen:v0n
  ri:il          = load:i0n
  rv:lon   = 0.0 = v0n
  end
NODE field
  rv:fgv   = 0.0 = v_reg:v0f = sgen:v0f
   v:fv    =       v_reg:v1f = sgen:v1f
  ri:fgi   =       v_reg:i0f = sgen:i0f
   i:fi    =       v_reg:i1f = sgen:i1f
   end
NODE Vreg
  rv:vbs   = 367.42 = v_reg:vbs
   v:vt    =          v_reg:vt
   v:ph    =          v_reg:phase
   end
!
```

Figure 5.3-4 v.init : Initialization File

```
!v.init
!Norbert H. Doerry
!
INITIALIZE
  v_reg:i0f    -38.499
  v_reg:i1f     38.499
  sgen:i0f     -38.499
  sgen:i1f      38.499
  sgen:psi_d     1
  sgen:s_wm      188.5
  sgen:eq_p      1
  sgen:eq_pp     1
  END
!
NODE VOLTAGE INITIALIZATIN
  C:v         367.4
  C:v_b      -183.7
  C:v_c      -183.7
  Mech:theta    0
  Mech:wm_dt    0
  ref:Te        0
  ref:Psi_q     0
  ref:Psi_d     1
  field:fv     52.56
  Vreg:vt     367.4
  Vreg:ph       0
  END
!
```

Figure 5.3-5 v.sim : Simulation File

```
!v.sim
!Norbert H. Doerry
!
SIMULATION
DISPLAY
  C:v
  C:v_b
```

```
  C:v_c
  sgen:Tepu
  sgen:vd
  sgen:vq
  sgen:id
  sgen:iq
  sgen:ifd
  Vreg:vt
  field:fv
  ref:Te
  Mech:wm
  load:Ia
  end
!
TIME_STEP  0.00025
TMIN       0
TMAX       8
PRINT_STEP 0.002
DELTA      0.01
DELTA_MIN  0.01
CONVERGE   1e-10
MAX_ITERATION 50
REFERENCE
  I : C:i
  END
EXTERNAL INPUTS
  sw:Switch 0 0
  sw:Switch 1 0.01
  sw:Switch 0 4.0
  END
!
```

The results of the simulation are shown in figures 5.3-6 through 5.3-11. Figure 5.3-6 shows the speed regulation of the governor. Notice the droop in frequency caused by the addition of the load (2.2% in this case). The frequency transient dies out within 4 seconds on both the application and removal of the load. The terminal voltage has interesting characteristics on both the addition and the removal of the load. On the application of the load, the terminal voltage experiences a negative voltage spike as the current builds up in the load inductance. On removal of the load, the network is in asymmetrical operation as each phase switch opens on the zero crossing of the current. Notice that the relatively high gain on the voltage regulator results in a small steady state error in the terminal voltage magnitude. Figure 5.3-8 shows the field voltage as generated by the voltage regulator. Even with the large change in load and large voltage regulator gain, the field voltage magnitude remains within the clipping limits.

Figure 5.3-6 RPM vs Time

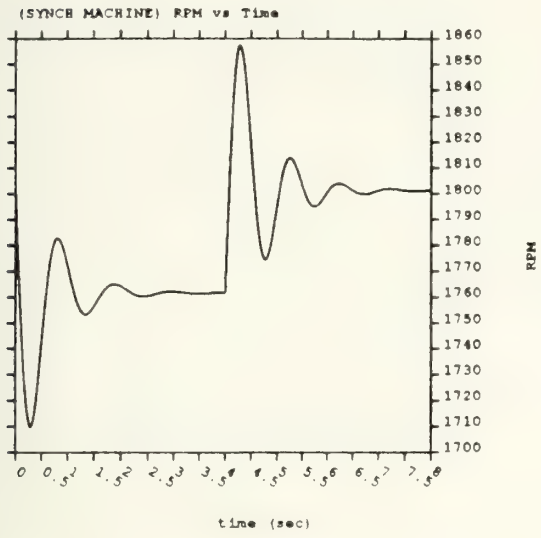


Figure 5.3-7 Terminal Voltage vs Time

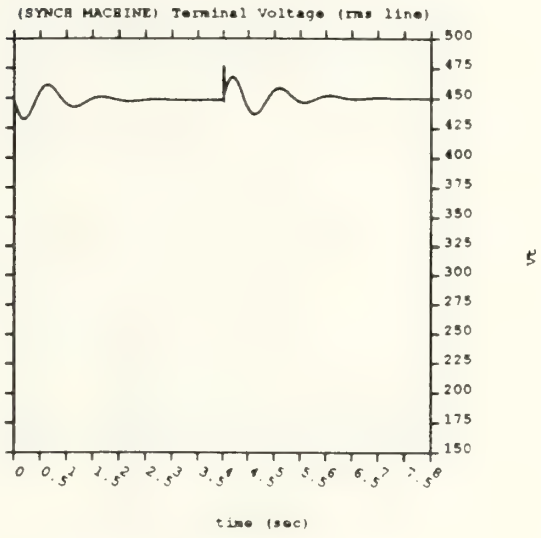


Figure 5.3-8 Field Voltage vs Time

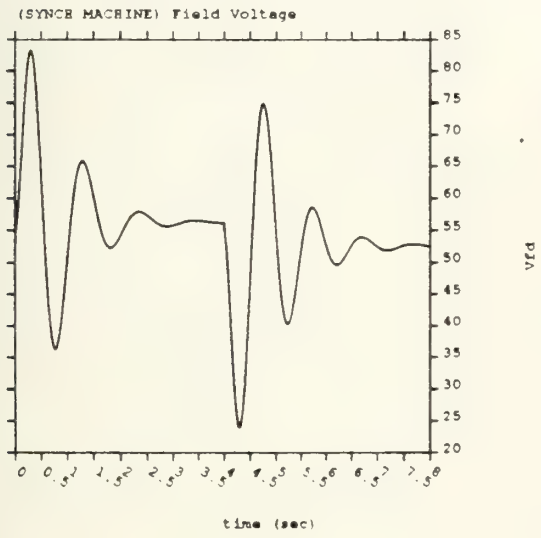


Figure 5.3-9 Tepu vs Time

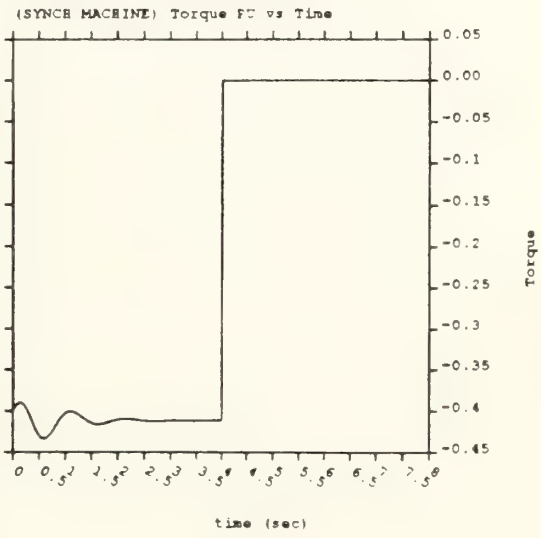


Figure 5.3-10 id and iq vs Time

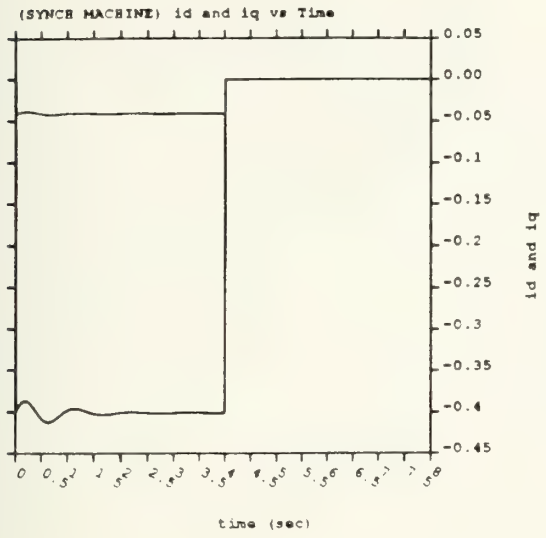
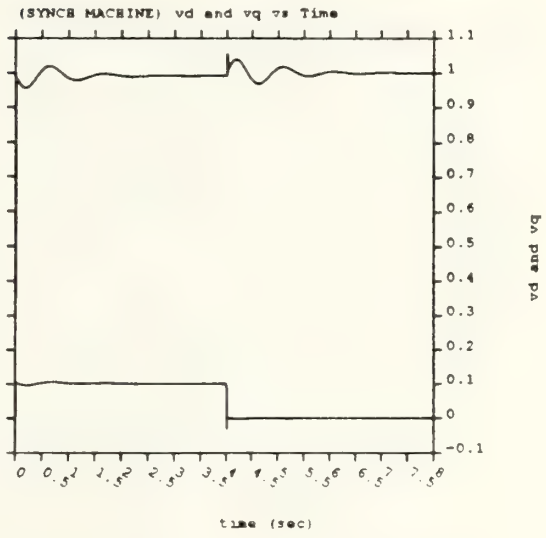


Figure 5.3-11 vd and vq vs Time



5.4 Synchronous Generator: Two Phase Fault

An electrical system casualty that can easily occur on shipboard systems is a two phase fault where two of the three lines of the distribution system are shorted together. This problem is difficult to solve analytically due to the asymmetrical nature of the resulting network. Figure 5.4-1 shows the SEPSIP network congifuration used to solve this problem numerically. The network is a modification of the network used in section 5.3. The transmission line element was added to prevent potential problems with the voltage regulator model. The voltage regulator model calculates the terminal voltage of the synchronous machine assuming a nonzero balanced voltage. If the terminal voltage of all three phases goes to zero, there is the possibity of the voltage regulator generating a singular Jacobian matrix. The transmission line assures a slightly positive voltage magnitude.

Figure 5.4-1 Synchronous Generator: Two Phase Fault

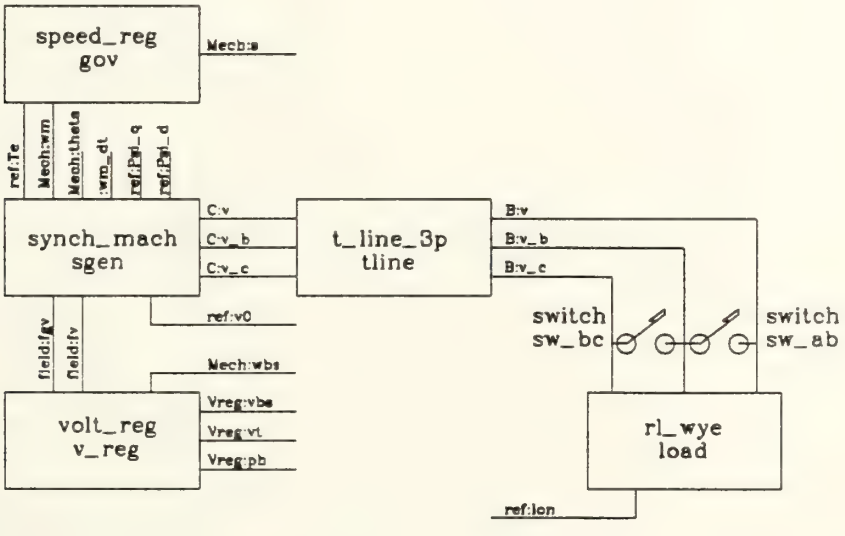


Figure 5.4-2 w.elm: Element Description File

```
! w.elm
! Norbert H. Doerry
!
synch_mach sgen
  xd        1.38
  xq        0.26
  xd_p      0.25
  xd_pp     0.171
```

```
  xq_pp     0.171
  xal       0.1
  Tdo_p     2.9
  Tdo_pp    0.0
  Tqo_pp    0.0
  Tad       0.09954
  Ifnl     38.5
  H         0.651
  pp        2.0
  wbs     376.99
  Vdb     367.4235
  Pbs       2.5e6
  end
rl_wye load
  R  0.2
  L  25e-6
  R1 1000
  end
t_line_3p tline
  R .00001
  L 0.0
  R1 1000.0
  end
switch sw_ab
  end
switch sw_bc
  end
speed_reg gov
  wnlo   187.36
  wds     31.69
  wdTepu -10.07
  TBS     13262.6
  Tg       0.3275
  B        0
  end
volt_reg v_reg
  Vfdbs   52.56
  K       20.0
  Tvr      0.15
  Vfmin    0
  Vfmax    120
  end
!
```

Figure 5.4-3 w.net: Network Description File

```
! w.net
! Norbert H. Doerry
!
NODE C
  v3:v = sgen:v0a = tline:v0a = v_reg:v0a
  i3:i = sgen:i0a = tline:i0a
  end
NODE B
  v:v   = tline:v1a = load:v0a = sw_ab:v0
  v:v_b = tline:v1b = load:v0b = sw_ab:v1 = sw_bc:v0
  v:v_c = tline:v1c = load:v0c = sw_bc:v1
  i:i   = tline:i1a = load:i0a = sw_ab:i0
  i:i_b = tline:i1b = load:i0b = sw_ab:i1 = sw_bc:i0
  i:i_c = tline:i1c = load:i0c = sw_bc:i1
  end
NODE Mech
  v:theta = sgen:theta
 rv:wbs = 3[illegible] = v_reg:wbs
  v:wm    = sgen:wm = gov:wm
  v:wm_dt = sgen:wm_dt
 rv:s   = .04 = gov:s
  end
NODE ref
  v:Te      = sgen:Te = gov:Tm
  v:Psi_q   = sgen:Psi_q
```

```
  v:Psi_d    = sgen:Psi_d
 rv:v0 = 0   = sgen:v0n
 ri:il       = load:i0n
 rv:lon = 0  = load:v0n
  end
NODE field
 rv:fgv  = 0.0 = v_reg:v0f = sgen:v0f
  v:fv   =       v_reg:v1f = sgen:v1f
 ri:fgl  =       v_reg:i0f = sgen:i0f
  i:fi   =       v_reg:i1f = sgen:i1f
  end
NODE Vreg
 rv:vbs  = 367.42 = v_reg:vbs
  v:vt   = v_reg:vt
  v:ph   = v_reg:phase
  end
!
```

Figure 5.4-4 w.init: Initialization File

```
! w.init
! Norbert H. Doerry
!
INITIALIZE
!
  v_reg:i0f         41.26
  v_reg:i1f        -41.26
  v_reg:vt         366.15
  v_reg:Verr         0.0034541
  v_reg:Vsig         0.069814
  v_reg:theta        0.12889
  gov:Tm_order       0.41106
  gov:Tmpu           0.41204
  sgen:i0a          -1697.5
  sgen:i0b           1442.6
  sgen:i0c            254.8
  sgen:i0f         -41.26
  sgen:i1f          41.26
  sgen:s_theta     -0.38396
  sgen:s_wm        184.49
  sgen:wm_dt         0.048494
  sgen:psi_d         1.0147
  sgen:psi_q        -0.10438
  sgen:eq_p          1.025
  sgen:eq_pp         1.0218
  sgen:ed_pp         0.035732
  sgen:d_psi_d      -0.00067673
  sgen:d_psi_q       2.29021e-05
  sgen:d_eq_p       -0.00066281
  sgen:d_eq_pp       0
  sgen:d_ed_pp       0
  tline:i0a          1697.5
  tline:i0b         -1442.6
  tline:i0c          -254.8
  tline:i1a         -1697.5
  tline:i1b          1442.6
  tline:i1c           254.8
  tline:ia          -1697.5
  tline:ib           1442.6
  tline:ic            254.8
  load:i0a           1697.5
  load:i0b          -1442.6
  load:i0c           -254.8
  load:va             339.49
  load:vb            -288.53
  load:vc             -50.957
  load:ia            1697.5
  load:ib           -1442.6
  load:ic            -254.8
END
!
```

```
NODE VOLTAGE INITIALIZATION
  C:v                339.49
  C:v_b             -288.53
  C:v_c              -50.957
  B:v                339.49
  B:v_b             -288.53
  B:v_c              -50.957
  Mech:theta         -0.38396
  Mech:wm            184.49
  Mech:wm_dt          0.048494
  ref:Te             5464.7
  ref:Psi_q          -0.10438
  ref:Psi_d           1.0147
  field:fv           56.229
  Vreg:vt            366.15
  Vreg:ph            -.51287
END
!
EXTERNAL INPUTS INITIALIZATION
  sw_ab:switch 0
  sw_bc:switch 0
  END
!
```

Figure 5.4-5 w2.sim: Simulation File

```
!w2.sim
!
SIMULATION
!
DISPLAY
  C:v
  C:v_b
  C:v_c
  sgen:Tepu
  sgen:vd
  sgen:vq
  sgen:id
  sgen:iq
  sgen:ifd
  Vreg:vt
  field:fv
  ref:Te
  Mech:wm
  load:Ia
  END
!
TIME_STEP  0.00025
TMIN       0
TMAX       4.5
PRINT_STEP 0.002
DELTA      0.01
DELTA_MIN  0.01
CONVERGE   1e-10
MAX_ITERATION 50
REFERENCE
  I:C:i
  END
EXTERNAL INPUTS
  sw_ab:switch  1 0.5
  sw_ab:switch  0 1.5
  END
!
```

The results of the simulation are shown in figures 5.4-6 through 5.4-11. Notice that the terminal voltage oscillates during the fault due to the method by which the voltage regulator determines the rms voltage. Figure 5.4-6 shows a high frequency oscillation of the generator

rotor during the fault. The field voltage clips at the maximum level which causes the terminal voltage to drop in magnitude during the fault. After the fault clears, the generator returns to the steady state condition within about three seconds.

Figure 5.4-6 RPM vs Time

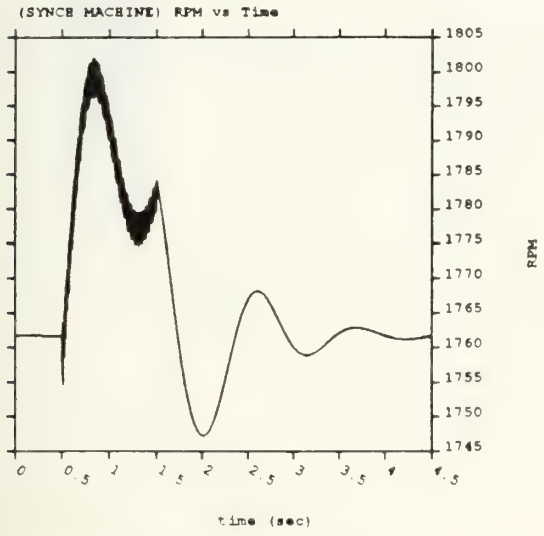


Figure 5.4-7 Torque PU vs Time

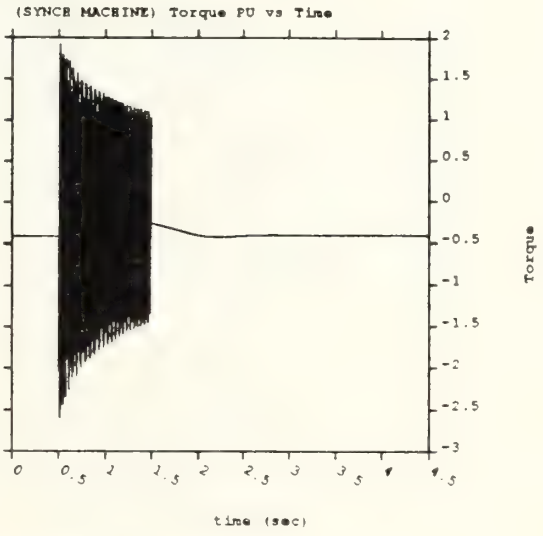


Figure 5.4-8 vd and vq vs Time

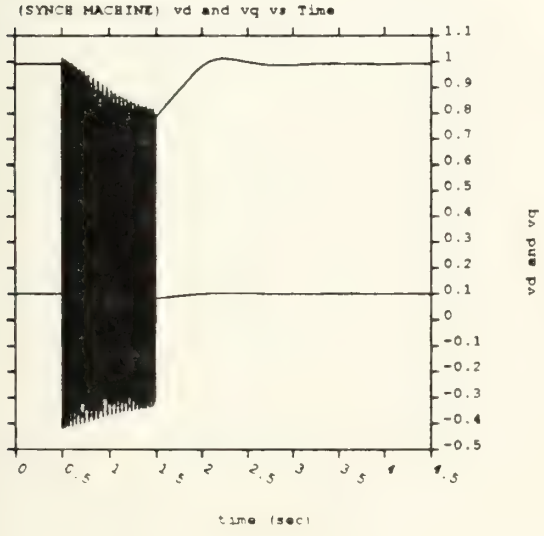


Figure 5.4-9 id and iq vs Time

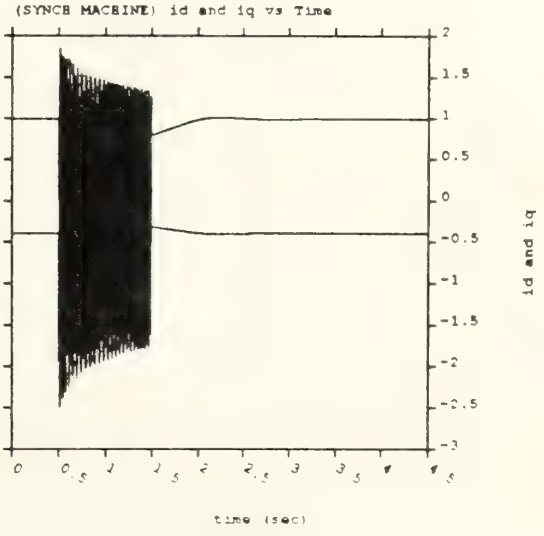


Figure 5.4-10 Line Voltage

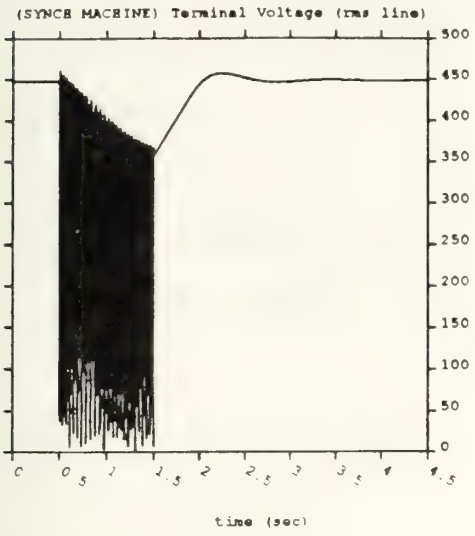
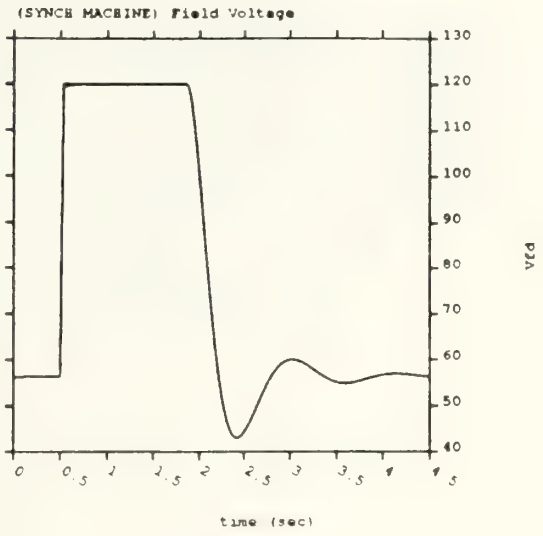


Figure 5.4-11 Field Voltage



5.5 Synchronous Generator: Three Phase Fault

The previous section simulated a two phase fault. This simulation shows the transient response of the synchronous generator due to a symmetrical three phase fault. The network is identical to the one used in section 5.4. The only change in the input file was the addition of two entries in the external input queue of the Simulation File.

Figure 5.5-1 Synchronous Generator: Three Phase Fault

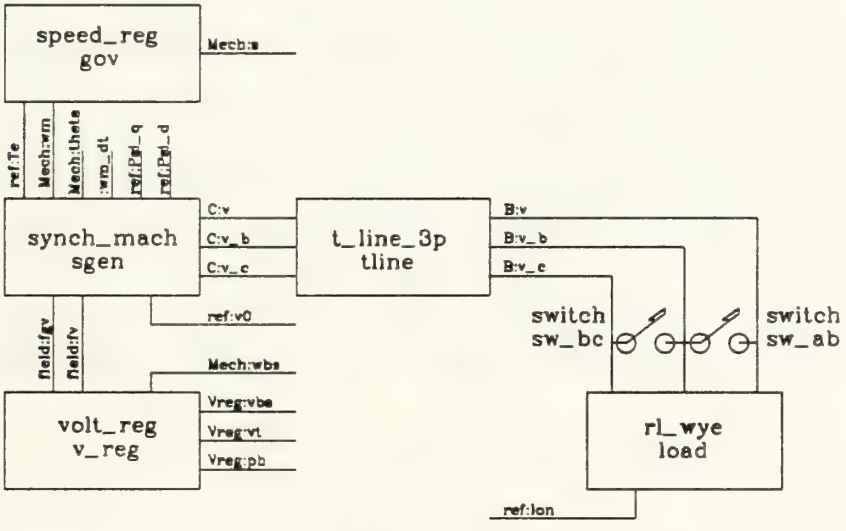


Figure 5.5-2 w3.sim: Simulation File

```
!w3.sim
!
SIMULATION
!
DISPLAY
  C:v
  C:v_b
  C:v_c
  sgen:Tepu
  sgen:vd
  sgen:vq
  sgen:id
  sgen:iq
  sgen:ifd
  Vreg:vt
  field:fv
  ref:Te
  Mech:wm
  load:Ia
  END
!
TIME_STEP   0.00025
TMIN        0
TMAX        4.5
```

```
PRINT_STEP 0.002
DELTA      0.01
DELTA_MIN  0.01
CONVERGE   1e-10
MAX_ITERATION 50
REFERENCE
  I:C:i
  END
EXTERNAL INPUTS
  sw_ab:switch  1 0.5
  sw_bc:switch  1 0.5
  sw_ab:switch  0 1.5
  sw_bc:switch  0 1.5
  END
!
```

The results of the simulation are shown in figures 5.5-3 through 5.5-8. Initially, the speed of the generator increases due to the sudden loss of load. This speed is controlled however, by the dynamics of the speed governor. During the short, the line voltage drops almost to zero. The voltage regulator responds to this by increasing the field voltage until clipping occurs at the preset value of 120 volts. The field voltage stays at this level well after the fault clears and until the line voltage approaches its reference. The torque characteristic is the typical response expected during a three phase fault.

Figure 5.5-3 RPM vs Time

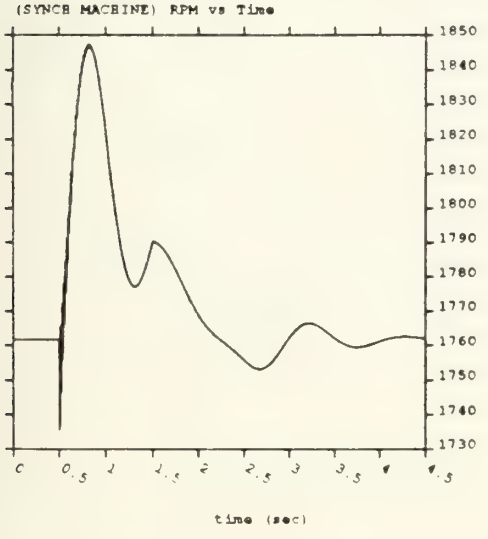


Figure 5.5-4 Torque PU vs Time

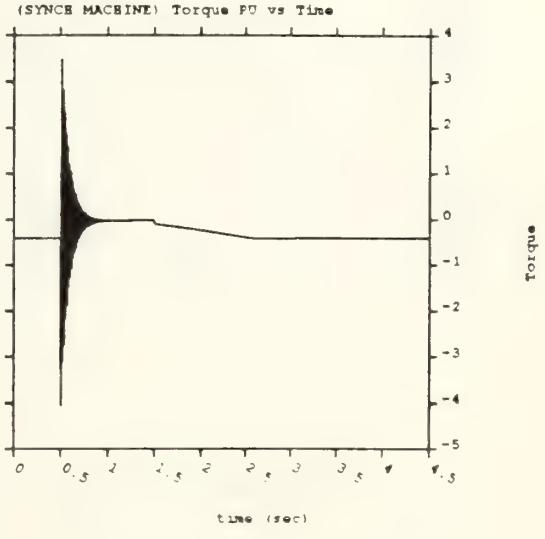


Figure 5.5-5 vd and vq vs Time

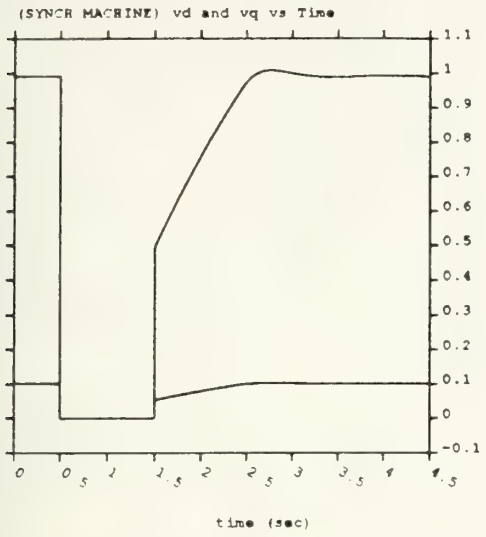


Figure 5.5-6 id and iq vs Time

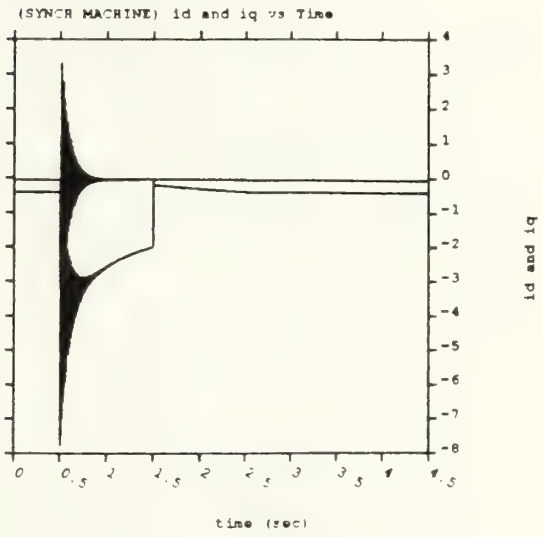


Figure 5.5-7 Line Voltage

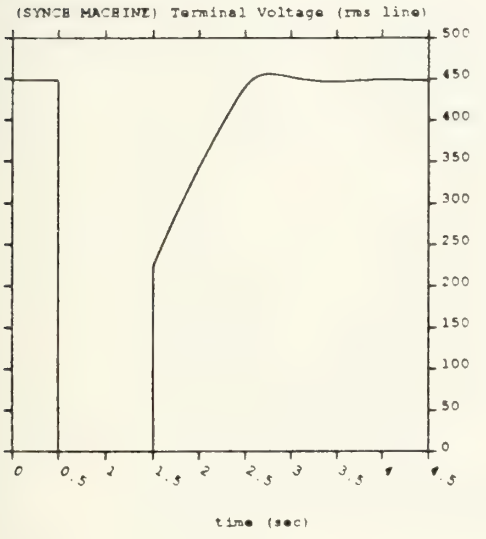
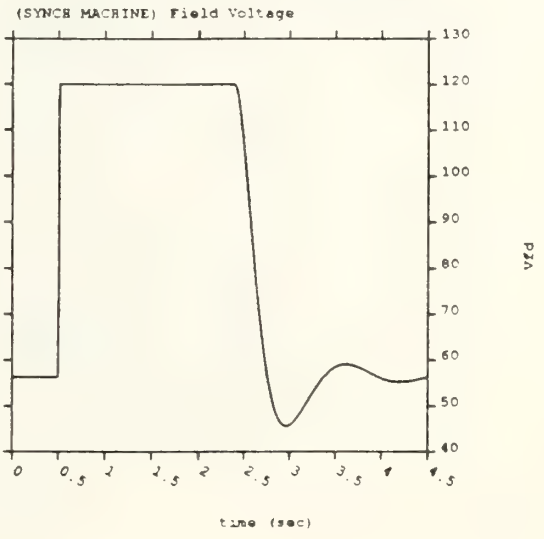


Figure 5.5-8 Field Voltage



5.6 Paralleled Synchronous Generators: Switched Load

Shipboard generators often operate paralleled with one another. This simulation shows the transient effects on two identical generators that are connected in parallel and subjected to a .4 per unit load for 4 seconds. The generators and load are identical to those used in section 5.3. Figure 5.6-1 shows the network structure. The transmission line element was inserted in parallel with the generator paralleling switch to prevent a singular system Jacobian matrix when the switch is opened. Without the transmission line, there would be two independent circuits when the switches are open and only one independent circuit with the switches closed. The problem stems from the requirement for one reference voltage subnode and one reference current subnode for each independent circuit. Adding a transmission line ensures there is always only one independent circuit. By assigning a large resistance to the transmission line, its effect on the solution is negligible.

Figure 5.6-1 Paralleled Synchronous Generators

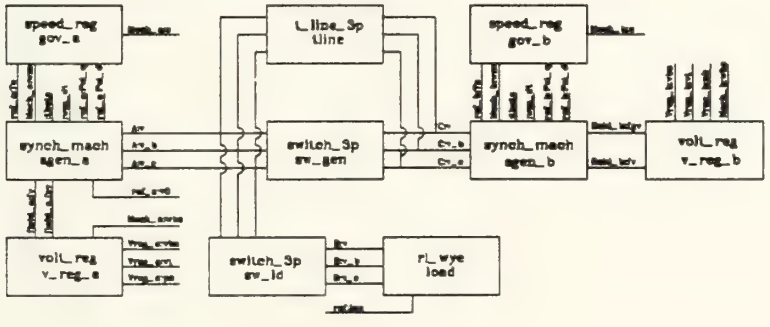


Figure 5.6-2 x.elm : Element Description File

```
! x.elm
! Norbert H. Doerry
!
synch_mach sgen_a
  xd          1.38
  xq          0.26
  xd_p        0.25
  xd_pp       0.171
  xq_pp       0.171
  xal         0.1
  Tdo_p       2.9
  Tdo_pp      0.0
  Tqo_pp      0.0
```

```
  Tad        0.09954
  Ifnl      38.5
  H          0.651
  pp         2
  wbs      376.99
  Vdb      367.4235
  Pbs      2.5e6
  end
!
synch_mach sgen_b
  xd         1.38
  xq         0.26
  xd_p       0.25
  xd_pp      0.171
  xq_pp      0.171
  xal        0.1
  Tdo_p      2.9
  Tdo_pp     0.0
  Tqo_pp     0.0
  Tad        0.09954
  Ifnl      38.5
  H          0.651
  pp         2
  wbs      376.99
  Vdb      367.4235
  Pbs      2.5e6
  end
!
speed_reg gov_a
  wnlo     187.36
  wds       31.69
  wdTepu   -10.07
  TBS       13262.6
  Tg         0.3275
  B          0.0
  end
!
speed_reg gov_b
  wnlo     187.36
  wds       31.69
  wdTepu   -10.07
  TBS       13262.6
  Tg         0.3275
  B          0.0
  end
!
volt_reg v_reg_a
  Vfdbs     52.56
  K         20.0
  Tvr        0.15
  Vfmax    120
  Vfmin      0
  end
!
volt_reg v_reg_b
  Vfdbs     52.56
  K         20.0
  Tvr        0.15
  Vfmax    120
  Vfmin      0
  end
!
rl_wye load
  R 6.2
  L 25e-6
  R1 1000
  end
!
switch_3p sw_ld
  end
switch_3p sw_gen
  end
```

```
!
t_line_3p tline
  R 100000
  R1 0
  L  0
  end
!
```

Figure 5.6-3 x.net : Network Description File

```
! x.net
!
NETWORK
!
NODE C
  v3:v = sgen_b:v0a = sw_gen:v0a = tline:v0a = v_reg_b:v0a
  i3:i = sgen_b:i0a = sw_gen:i0a = tline:i0a
  end
NODE B
  v3:v = sw_ld:v1a = load:v0a
  i3:i = sw_ld:i1a = load:i0a
  end
NODE A
  v3:v = sgen_a:v0a = sw_ld:v0a = sw_gen:v1a = tline:v1a = v_reg_a:v0a
  i3:i = sgen_a:i0a = sw_ld:i0a = sw_gen:i1a = tline:i1a
  end
NODE Mech_a
   v:theta     = sgen_a:theta
  rv:wbs = 377 = v_reg_a:wbs
   v:wm        = sgen_a:wm = gov_a:wm
   v:wm_dt     = sgen_a:wm_dt
  rv:s = 0.04  = gov_a:s
   end
NODE Mech_b
   v:theta     = sgen_b:theta
  rv:wbs = 377 = v_reg_b:wbs
   v:wm        = sgen_b:wm = gov_a:wm
   v:wm_dt     = sgen_b:wm_dt
  rv:s = 0.04  = gov_b:s
   end
NODE ref_a
   v:Te        = sgen_a:Te = gov_a:Tm
   v:Psi_q     = sgen_a:Psi_q
   v:Psi_d     = sgen_a:Psi_d
  rv:v0 = 0.0  = sgen_a:v0n
   end
NODE ref_b
   v:Te        = sgen_b:Te = gov_b:Tm
   v:Psi_q     = sgen_b:Psi_q
   v:Psi_d     = sgen_b:Psi_d
  rv:v0 = 0.0  = sgen_b:v0n
   end
NODE ref_ld
  ri:il = load:i0n
  rv:lon = 0.0 = load:v0n
  end
NODE field_a
  rv:fgv = 0.0 = v_reg_a:v0f = sgen_a:v0f
   v:fv  =       v_reg_a:v1f = sgen_a:v1f
  ri:fgi =       v_reg_a:i0f = sgen_a:i0f
   i:fi  =       v_reg_a:i1f = sgen_a:i1f
   end
NODE field_b
  rv:fgv = 0.0 = v_reg_b:v0f = sgen_b:v0f
   v:fv  =       v_reg_b:v1f = sgen_b:v1f
  ri:fgi =       v_reg_b:i0f = sgen_b:i0f
   i:fi  =       v_reg_b:i1f = sgen_b:i1f
   end
NODE Vreg_a
  rv:vbs = 367.42 = v_reg_a:vbs
```

```
   v:vt  = v_reg_a:vt
   v:ph  = v_reg_a:phase
   end
NODE Vreg_b
  rv:vbs = 367.42 = v_reg_b:vbs
   v:vt  = v_reg_b:vt
   v:ph  = v_reg_b:phase
   end
!
```

Figure 5.6-4 x.init : Initialization File

```
! x.init
INITIALIZE
!
  v_reg_a:i0f       38.499
  v_reg_a:i1f      -38.499
  sgen_a:i0a         0
  sgen_a:i0b         0
  sgen_a:i0c         0
  sgen_a:i0f       -38.499
  sgen_a:i1f        38.499
  sgen_a:s_theta     0.0
  sgen_a:s_wm      188.5
  sgen_a:s_wm_dt     0
  sgen_a:psi_d       1
  sgen_a:psi_q       0
  sgen_a:eq_p        1
  sgen_a:eq_pp       1
  sgen_a:ed_pp       1
  gov_a:Tm_order     0
  gov_a:Tmpu         0
!
  v_reg_b:i0f       38.499
  v_reg_b:i1f      -38.499
  sgen_b:i0a         0
  sgen_b:i0b         0
  sgen_b:i0c         0
  sgen_b:i0f       -38.499
  sgen_b:i1f        38.499
  sgen_b:s_theta     0.0
  sgen_b:s_wm      188.5
  sgen_b:s_wm_dt     0
  sgen_b:psi_d       1
  sgen_b:psi_q       0
  sgen_b:eq_p        1
  sgen_b:eq_pp       1
  sgen_b:ed_pp       1
  gov_b:Tm_order     0
  gov_b:Tmpu         0
  END
!
NODE VOLTAGE INITIALIZATON
  C:v          367.4
  C:v_b       -183.7
  C:v_c       -183.7
  A:v          367.4
  A:v_b       -183.7
  A:v_c       -183.7
  Mech_a:theta  0
  Mech_a:wm_dt  0
  Mech_a:wm    188.5
  ref_a:Te       0
  ref_a:Psi_q    0
  ref_a:Psi_d    1
  field_a:fv    52.56
  Vreg_a:vt    367.4
  Vreg_a:ph      0.0
  Mech_b:theta  0
  Mech_b:wm_dt  0
  Mech_b:wm    188.5
```

```
  ref_b:Te       0
  ref_b:Psi_q    0
  ref_b:Psi_d    1
  field_b:fv    52.56
  Vreg_b:vt    367.4
  Vreg_b:ph      0.0
  END
!
EXTERNAL INPUTS INITIALIZATION
  sw_ld:Switch  0
  sw_gen:Switch 0
  END
!
```

Figure 5.6-5 x2.sim : Simulation File

```
!x2.sim
!
SIMULATION
!
DISPLAY
  A:v
  A:v_b
  A:v_c
  sgen_a:Tepu
  sgen_a:vd
  sgen_a:vq
  sgen_a:id
  sgen_a:iq
  sgen_a:ifd
  Vreg_a:vt
  field_a:fv
  ref_a:Te
  Mech_a:wm
  load:Ia
  sgen_b:Tepu
  sgen_b:vd
  sgen_b:vq
  sgen_b:id
  sgen_b:iq
  sgen_b:ifd
  Vreg_b:vt
  field_b:fv
  ref_b:Te
  Mech_b:wm
  END
!
TIME_STEP    0.00025
TMIN         0
TMAX         8
PRINT_STEP   0.002
DELTA        0.01
DELTA_MIN    0.01
CONVERGE     1e-10
MAX_ITERATION  50
REFERENCE
  I:A:i
  END
!
EXTERNAL INPUTS
!
  sw_gen:Switch 1 0.004
  sw_ld:Switch 0 0.0
  sw_ld:Switch 1 0.01
  sw_ld:Switch 0 4.0
  END
!
```

The results of the simulation are shown in figures 5.6-6 through 5.6-11. Since the generators are always in parallel when the load switches on and off, the response is similar to the results in section 5.3 but with smaller deviations due to the addition of the second generator.

Figure 5.6-6 RPM vs Time

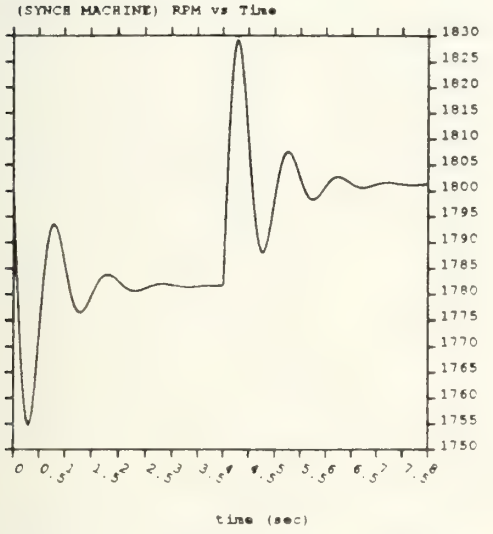


Figure 5.6-7 Terminal Voltage vs Time

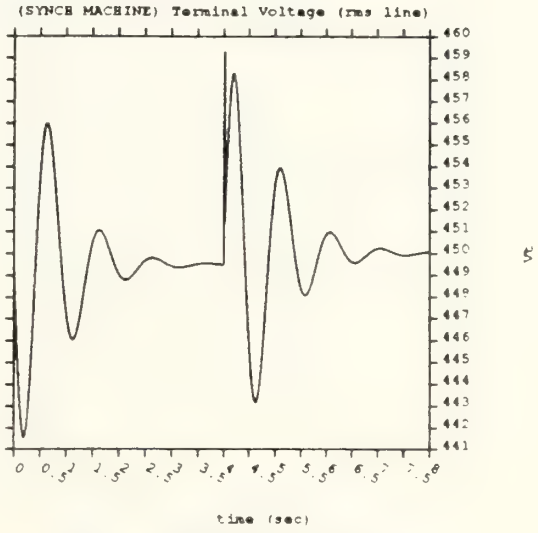


Figure 5.6-8 Field Voltage vs Time

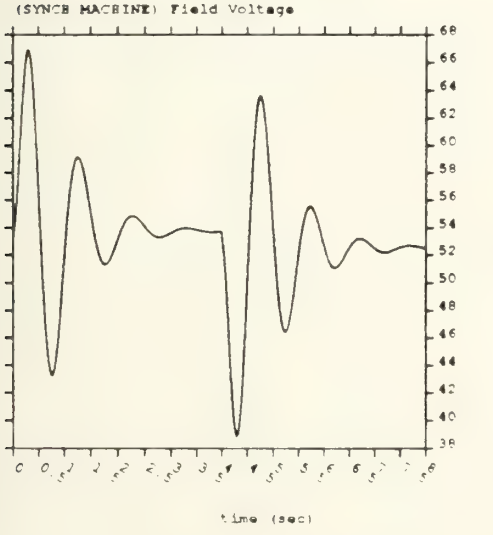


Figure 5.6-9 Tepu vs Time

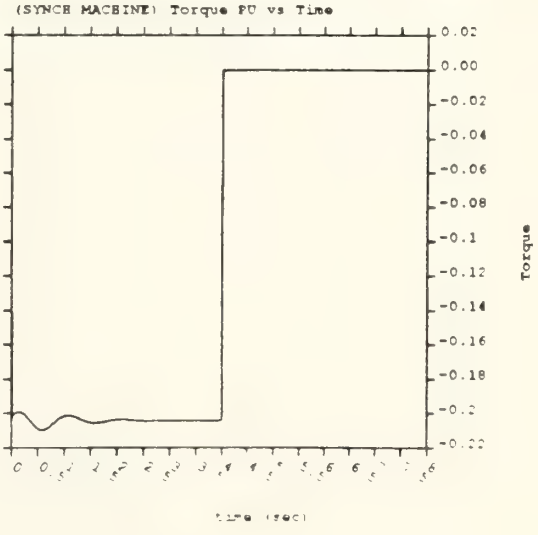


Figure 5.6-10 id and iq vs Time

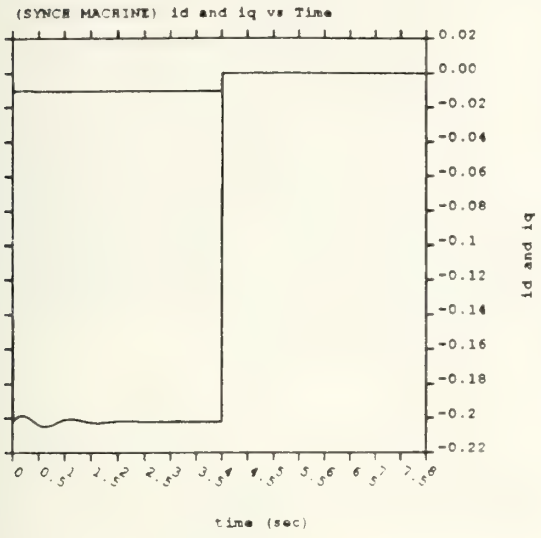
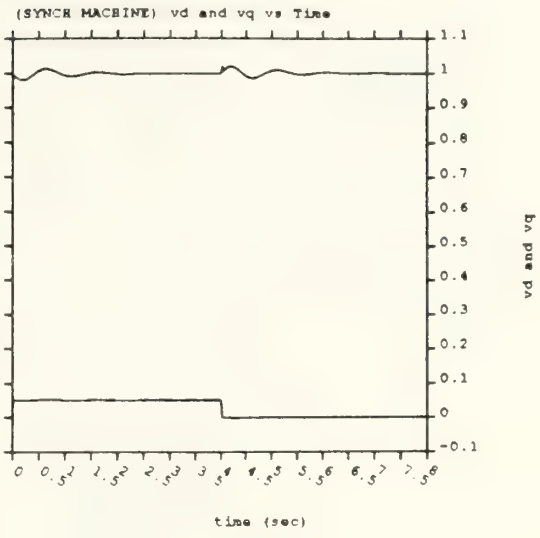


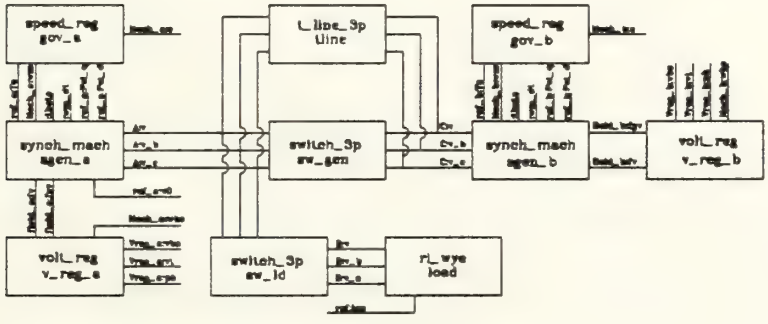
Figure 5.6-11 vd and vq vs Time



5.7 Paralleled Synchronous Generators: Switched Load

Shipboard generators often operate paralleled with one another. This simulation shows the transient effects on two identical generators that are connected in parallel and subjected to a .4 per unit load for 4 seconds. After the 4 seconds, one of the generators is detached from the load. The generators and load are identical to those used in section 5.6. Figure 5.7-1 shows the network structure which is identical to the structure in the previous section.

Figure 5.7-1 Paralleled Synchronous Generators



The input file for this simulation is identical to the input file described in section 5.6 with the exception of minor changes to the simulation file.

Figure 5.7-2 x3.sim : Simulation File

```
!x3.sim
!
SIMULATION
!
DISPLAY
  A:v
  A:v_b
  A:v_c
  sgen_a:Tepu
  sgen_a:vd
  sgen_a:vq
  sgen_a:id
  sgen_a:iq
  sgen_a:ifd
  Vreg_a:vt
  field_a:fv
  ref_a:Te
  Mech_a:wm
  load:Ia
  sgen_b:Tepu
  sgen_b:vd
  sgen_b:vq
  sgen_b:id
  sgen_b:iq
```

```
  sgen_b:ifd
  Vreg_b:vt
  field_b:fv
  ref_b:Te
  Mech_b:wm
  END
!
TIME_STEP    0.00025
TMIN         0
TMAX         8
PRINT_STEP   0.002
DELTA        0.01
DELTA_MIN    0.01
CONVERGE     1e-10
MAX_ITERATION  50
REFERENCE
  I:A:i
  END
!
EXTERNAL INPUTS
!
  sw_gen:Switch 0 0.0
  sw_gen:Switch 1 0.004
  sw_ld:Switch  0 0.0
  sw_ld:Switch  1 0.01
  sw_gen:Switch 0 4.0
  END
!
```

The results of the simulation are shown in figures 5.7-3 through 5.7-8. For the first four seconds, the simulation is identical to the previous simulation. When the switch connecting the two generators opens however, the two generators have different transient responses as they attain their new steady state conditions. Figure 5.7-3 clearly shows the effect of the droop characteristic of the speed governor.

Figure 5.7-3 RPM vs Time

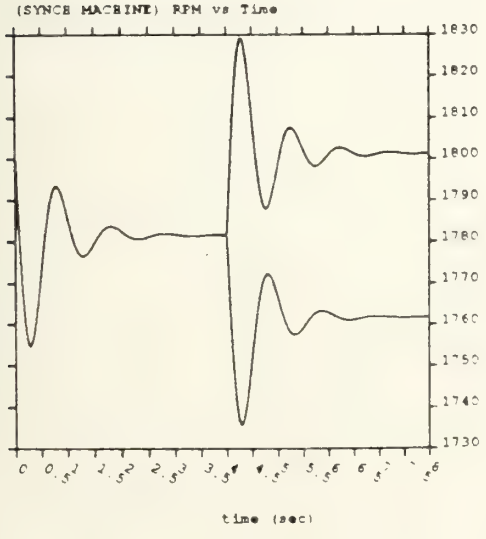


Figure 5.7-4 Terminal Voltage vs Time

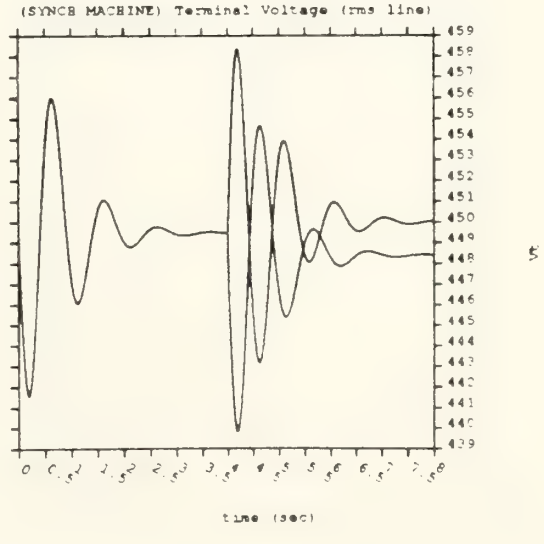


Figure 5.7-5 Field Voltage vs Time

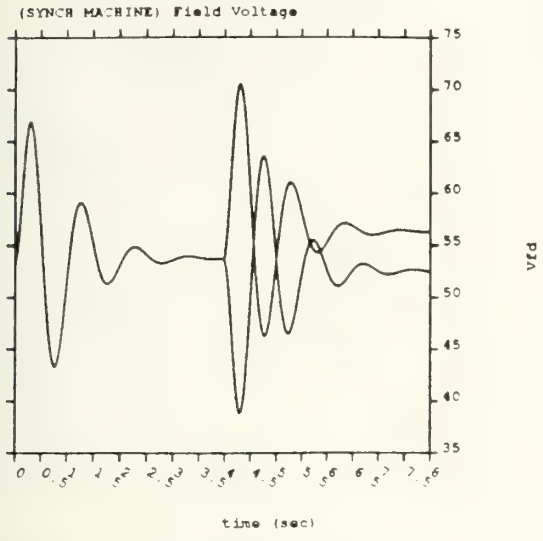


Figure 5.7-6 Tepu vs Time

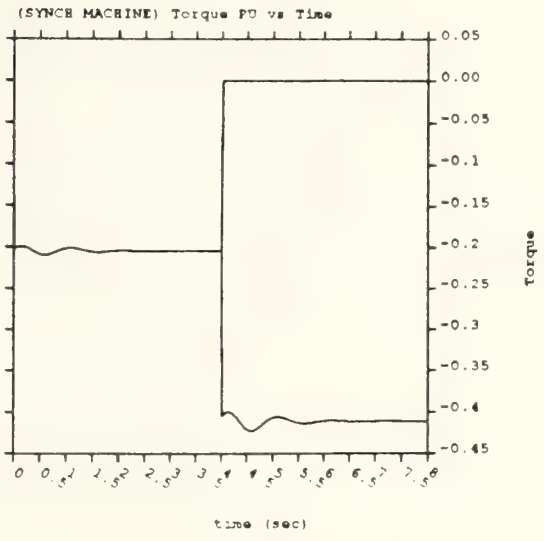


Figure 5.7-7 id and iq vs Time

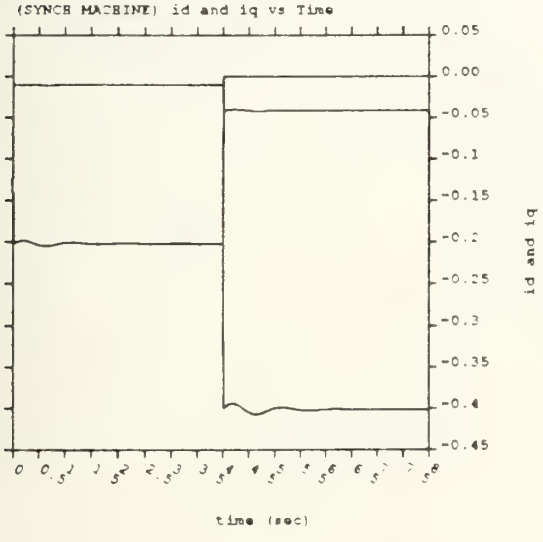
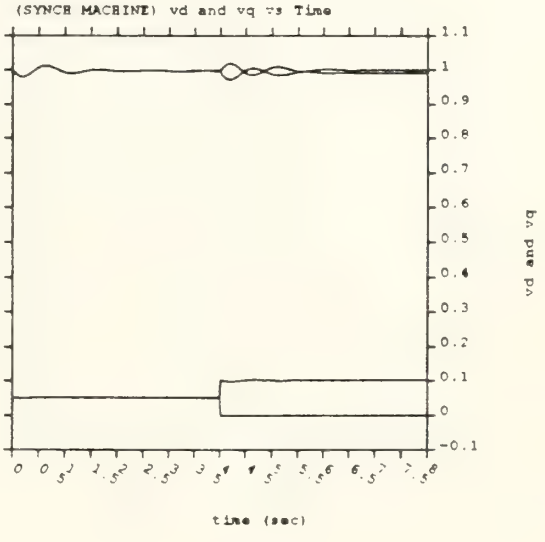


Figure 5.7-8 vd and vq vs Time



CHAPTER 6

CONCLUSIONS

6.1 Assessment of SEPSIP

In its present form, SEPSIP is useful for analyzing simple and moderately complex electrical networks. Since the time required for a simulation is proportional to the cubic of the order of the system (Gaussian Elimination), large simulations require an unreasonable amount of time. Careful design of the network to minimize the system order can help tremendously in reducing the execution time of SEPSIP. Other possible methods of improving the speed of SEPSIP include running the program on a faster computer and making simple changes to the program. The following section provides details on several options for increasing SEPSIP's speed.

The decision to use SEPSIP or a general simulation language such as ACSL for the simulation of shipboard systems depends on what exactly is desired to be modeled. If the desire is to simulate the respose of a new type of device, ACSL may be superior. Writing robust device driver routines for SEPSIP that are fast, stable and accurate can be difficult and time intensive. Creating the device models on ACSL should normally require less time since physically writing and compiling code is not required. Once the device models have been created however, the advantage shifts greatly to SEPSIP. SEPSIP's built in provisions for organizing and interconnecting a number of devices into a network are far superior to the capabilities resident in ACSL. ACSL is not really designed to handle network architectures and forcing it to do so results in large unmanageable input files that are hard to debug. In short, ACSL is better at modeling individual devices while SEPSIP is far superior in organizing models into networks.

6.2 Future Improvements

SEPSIP, like any other computer program, will never be completed. There will always be a number of possible improvements that only take time and effort to impliment. The following sections detail some of the possible improvements to the present version of SEPSIP that would greatly extend its utility.

6.2.1 Variable Time Step

Many simulations would run much faster if a variable time step were incorporated. Presently, the time step must be small enough to capture the transient response of the fastest time constant of interest. However, since the fast time constant is only important for a small time period after a disturbance, using the small time step for the remaining time is very inefficient. When only slow time constants are important, larger time steps can be used.

One way of implementing variable time steps in SEPSIP would be to include the option of adding the time step variable to the **EXTERNAL INPUT** subsection of the **SIMULATION** section. In this manner the time step variable, its new value, and the time that the new value takes effect can all be specified.

6.2.2 Replace Gaussian Elimination

The Newton-Raphson method used to solve the system of nonlinear equations requires the solution of the linear equation $\bar{J}\bar{x} = \bar{y}$. Presently, SEPSIP uses Gaussian Elimination with partial pivoting to solve for $\bar{x}$. For an **nth** order system, the number of floating point multiplication operations is proportional to $\mathbf{n}^3$. Since $\bar{x}$ is only used to produce a correction for the Newton-Raphson method, there is no need to solve exactly for it. An iterative method for solving for $\bar{x}$ can cut down the computation time considerably. The Gauss-Seidel method for example, requires on the order of only $\mathbf{n}^2$ multiplication operations for each iterations. If the number of iterations can be held well below the order of the system, a speed gain will be realized.

Giving the operator the choice of which method to use would be easy to implement and would not add any extra time to the execution of the simulation. This way the operator can experiment to determine which is the better method for the simulation under study.

6.2.3 Reuse of Jacobian Matrix

SEPSIP presently recalculates and inverts the Jacobian matrix for each iteration of every time step. If the Jacobian matrix does not change very fast, there is no need to recalculate it for each iteration. Allowing the user to specify how many times the Jacobian matrix is used before it is recalculated would greatly increase the speed of the program for certain simulations.

6.2.4 Output Variables and Output Subnodes

One way to increase the speed of a simulation is to decrease the order of the system by incorporating **output** variables and **output subnodes**. An output variable would be explicitly defined by an element and connected to one or more input variables through the output subnode. The output subnode would equate all of the input variables to the output variable. For this reason, there would not be a system variable associated with the output subnode. Presently, all the variables would have to be implicitly defined and connected with a voltage subnode that adds a system variable.

6.2.5 Action Files

Action Files are text files that contain a list of SEPSIP commands. When an Action File is called from within SEPSIP, control would pass from keyboard entry to the commands contained in the file. Once all of the commands have been executed, control shifts back to keyboard entries. This feature allows one to run a number of long simulations sequentially without reamaining at the computer terminal.

Modifying SEPSIP to accomodate Action Files would require replacing all calls to the system function **gets** with a call to another routine that would determine which input stream to use. The code would look similar to:

```
#include <stdio.h>
#include "doerry.h"

/* structure for keeping track of the present stream */
typedef struct Strm
{
  FILE *in;
  struct Strm *last;
}
STRM;

/* This is the initialization code which sets the
  stream to stdin */

static STRM *strm;

init_strm()
{
  char *calloc();

  strm = (STRM *) calloc((unsigned) 1 , sizeof(STRM));
  strm->in = stdin;
  strm->last = NULL;
}
/* edit_strm sets the current stream to the value passed
  to it */

edit_strm(stm)
FILE *stm;
{
  char *calloc();
  STRM *temp;

  temp = (STRM *) calloc((unsigned) 1, sizeof(STRM));
  temp->in = stm;
  temp->last = strm;
  strm = temp;
}
/* gets_in() replaces gets() as the routine for reading in a
line from stdin */

char *gets_in(string)
char *string;
{
  int i;

  i = fgets(string,MAXCHAR,strm->in);

  /* see if read to the end of the file */
  while (i == NULL && strm->last != NULL)
     {
        strm = strm->last;
        i = fgets(string,MAXCHAR,strm->in);
     }
  /* get rid of trailing carriage return */
  i = strlen(string) - 1;
  if (i >= 0 && string[i] == '\n') string[i] = NULL;

  return string;
}
```

6.2.6 Integrated Graphics

SEPSIP was intentionally written without any graphics to improve the portability of the code. In general, graphic subroutines tend to be machine specific and therefore require much revision when transferred to another computer. SEPSIP has circumvented this problem somewhat by using a totally separate plotting program (Norplot) to display the data. Unfortunately, the data can only be plotted once the simulation is totally finished. It would at times be beneficial to observe the display variables in a graphical form as the simulation progresses. If the simulation goes unstable, the program can immediately be halted instead of waiting for its completion.

Portability could still be maintained to a degree by using only a few general routines that could be easily modified for whatever system SEPSIP is installed on. These routines would include

terminit()	Initialize the Graphics Screen
clrscrn()	Clear the Graphics Screen
move(x,y)	Move to (x,y)
draw(x,y)	Draw to (x,y)
windset(xmn,xmx,ymn,ymx)	Set the coordinates of the corners
windget(xmn,xmx,ymn,ymx)	Get the coordinates of the corners
g_puts(str,height,rot)	Print a string on the Graphics Screen
termend()	End the Graphics Screen

These routines should all be contained in one file that can rewritten to conform to the graphics interface of each particular machine.

6.2.7 Implement external variable 'types'

The original concept for SEPSIP was to be able to specify the external input and output variables to be one of four types: floating point, integer, Boolean, and Switch. Presently, only the floating point type is truly implemented. Adding the capability to recognize and print out the four types would improve the ability of the user to specify his or her desires and to understand the results of the simulation. A switch being **on** is more informative than a switch having a value of 1.0. Likewise a variable called **overspeed_sensor** having a value of **false** is more informative than having a value of 0.0.

6.2.8 Optimization for speed

While a conscious effort was made to write efficient code, there is still plenty of room for improvement. There are a number of places where speed was sacrificed for clarity in reading the source code. In any software project, there is always the conflict between writing fast routines, and routines that can be easily understood. Usually it is a good idea to initially write understandable code that can easily be debugged. If the resulting code runs too slowly, the code can always be optimized. SEPSIP is presently at this state and requires some optimization to improve its performance.

6.2.9 Check for Recursive INCLUDE Files

The present version of SEPSIP does not check for recursive INCLUDE statements where a file tries to include itself or a file above itself in the stack of include files. Consequently, it possible to construct an input file that will cause SEPSIP to enter an infinite loop as it recursively opens the same files over and over again. Normally this is not a problem since for clarity the input file structure should not be very complex and any recursion would therefore be easily noticed. However, this condition should not be allowed to exist and SEPSIP should check for it.

6.2.10 Break Key

Once a simulation begins in the present version of SEPSIP, the only way to terminate the simulation before it is completed is to terminate the execution of the program with a "control c". Since a simulation can take some time to complete, it would be useful at times to be able to stop a simulation in the middle by depressing a specific key, observe some of the internal variables, and possibly continue the simulation. Unfortunately, there is no

standard way in the C programming language to determine if a key has been depressed without the user entering a carriage return[1]. Most systems do provide a way of accomplishing such "unbuffered IO" but the implementations are very system dependent.

1 C normally employs buffered input where the characters a person types in from the keyboard are not available to the program until the **return** key is depressed. This allows the user to edit the input line before the program has access to it. A consequence of buffered input is that when a program requests input from the keyboard, the program waits until a **return** is entered. In this case, we do not want the program to wait for a **return** because the operator would have to hit the **return** key every time the program checks for a character.

References

[1] Advanced Continuous Simulation Language (ACSL) User Guide / Reference Manual, Mitchell and Gauthier Associates, Concord, MA 1975.

[2] Basler Electric Company, Instruction Manual for Voltage Regulator Models SR4A & SR8A, Publication 9 0177 00 990, Revision B of July 1986.

[3] Calovic, M. S. and V. C. Strezoski, "Calculation of Steady-State Load Flows Incorporating System Control Effects and Consumer Self-Regulation Characteristics", Electrical Power & Energy Systems, Vol 3 No 2, April 1981, pages 65-74.

[4] Carlsen, K., E. H. Lenfest, J. J. LaForest, "MANTRAP Machine and Network Transients Program", 1976 Power Industry Computer Applications Conference, pages 144-150.

[5] Crandall, Stephen H., Engineering Analysis, A Survey of Numerical Procedures, Robert E. Krieger Publishing Company, Malabar, FL, 1986.

[6] Dalton, R.C., Turbine Generator Simulations for DD-692 Class 450 KW Machine and SSN-637 Class 200 KW Machine, Naval Ship System Engineering Station, Philadelphia, (NAVSSES Project C-267), March 1984.

[7] DD-963 Class Propulsion Plant Manual, Volume II, Chapter 11, Naval Sea Systems Command, United States Navy.

[8] Del Toro, Vincent, Electric Machines and Power Systems, Prentice-Hall, Inc., Englewood CLiffs, NJ, 1985.

[9] Diao, Yi-min, O. Wasynczuk, P. C. Krause, "Solution of State Equations in Terms of Separated Modes with Applications to Synchronous Machines", IEEE report 85-SM 346-2.

[10] Doerry, Norbert H., "Shipboard Electrical Generator Simulation", Term paper for 6.238, MIT, May 1988.

[11] Dommel, H. W., and N. Sato, "Fast Transient Stability Solutions", paper T72 137-3 presented at the IEEE Winter Meeting, New York, NY, January 30 - February 4, 1972.

[12] Fitzgerald, A. E., Charles Kingsley Jr., and Stephen D. Umans, Electric Machinery, 4th Edition, McGraw-Hill, Inc., 1983.

[13] Fröberg, Carl-Eric, Numerical Mathematics, Theory and Computer Applications, The Benjamin / Cummings Publishing Company, Inc., 1985.

[14] Hildebrand, F. B., Introduction to Numerical Analysis, McGraw-Hall, Inc., 1956.

[15] Kernighan, Brian W., and Dennis M. Ritchie, The C Programming Language, 2nd ed., Prentice Hall, Englewood Cliffs, NJ, 1988.

[16] Kirtley, J. L., Synchronous Machine Dynamic Models, LEES Technical Report TR-87-008, Massachusetts Institute of Technology, June 5, 1987.

[17] Krause, Paul C., Analysis of Electric Machinery, McGraw-Hill, Inc., 1986.

[18] Krause, Paul C., "Modeling of Shipboard Electric Power Distribution System", SBIR Phase I Final Report, P. C. Krause and Associates, Inc., West Lafayette, IN, July 1988.

[19] Larsen, E. V., and W. W. Price, "MANSTAB/POSSIM Power System Dynamic Analysis Programs - A New Approach Combining Nonlinear Simulation and Linearized State-Space/Frequency Domain Capabilities", 1977 Power Industry Computer Applications Conference, pages 350-357.

[20] Leonhard, W., Control of Electrical Drives, Springer-Verlag, Berlin, Heidelberg, 1985.

[21] Luini, James F., Richard P. Shulz and Anne E. Turner, "A Digital Computer Program for Analyzing Long Term Dynamic Response of Power Systems".

[22] Meyer, W. Scott, "Machine Translation of an Electromagnetic Transients Program (EMTP) Among Different Digital Computer Systems", 1977 Power Industry Computer Applications Conference, pages 272-277.

[23] Okamura, M., Y. O-oura, S. Hayashi, K. Uemura, and F. Ishiguro, "A New Power Flow Model and Solution Method - Including Load and Generator Characteristics and Effects of System Contorl Devices", IEEE Transactions of Power Apparatus and Systems, Vol. PAS-94 No. 3, May/June 1975, pages 1042-1049.

[24] Prasad, N. R., and R. D. Dunlop, "Three Phase Simulation of the Dynamic Interaction Between Synchronous Generators and Power Systems Using the Continuous Systems Modelling Program (CSMP III)", 1979 Power Industry Computer Applications Conference, pages 29-36.

[25] Rabinowitz, Philip, Numerical Methods for Nonlinear Algebraic Equations, Gordon and Breach Science Publishers, Londoon, 1970.

[26] Rowen, W. I., "Simplified Mathematical Representations of Heavy-Duty Gas Turbines", Journal of Engineering for Power, October 1983, Vol 105., pages 865-869.

[27] Sarma, Mulukutla, Synchronous Machines (Their Theory, Stability, and Excitation Systems), Gordon and Breach Science Publishers, New York, 1979.

[28] Sauer, P. W., and M. A. Pai, Course Guide and Notes for Power System Dynamics and Stability, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, Ill.

[29] Student Guide for DD-963 Engineering Division Officer, A-4H-0055, Surface Warfare Officers School Command, U.S. Navy, Newport, RI, April 1983.

[30] Stypulkowski, A., and E. Pollak, "Microprocessor Based Real-Time Simulation of a Multiple Gas Turbine Generator Electric Plant for Embedded Training", presented at the Society of Naval Architecture and Marine Engineers Spring Meeting STAR Symposium, Norfolk, VA May 21-24 1985.

[31] Velez-Reyes, Miguel and George C. Verghese, "Developing Reduced Order Electrical Machine Models Using Participation Factors", 12th IMACS World Congress, Paris, July 1988.

[32] Wood, Allen J., and Bruce F. Wollenberg, Power Generation Operation & Control, John Wiley and Sons, 1984.

[33] Woodson, Herbert H. and James R. Melcher, Electromechanical Dynamics, John Wiley and Sons, 1968.

[34] Woodward Governor Company, The Control of Prime Mover Speed, Manual 25031, 1981.

[35] Woodward Governor Company, Electrical Generating Systems, Synchronizing and Methods of Controlling Output, Manual 25104B, 1985.

[36] Woodward Governor Company, 9900-326 System, 9900-323 Electronic Governor Control, 9900-322 Interface Panel includes 9900-305 Temperature Sensor, Part 2, Theory of Operation and Calibration Procedures for Allison 501K-17 Gas-Turbine Engines, Manual 83029 Part 2B.

APPENDIX A

GLOSSARY

BALANCING THE SYSTEM

Balancing the System refers to the process of varying the input variables until all of the implicit variables for all of the elements are within tolerable limits of zero. When the Mean Square error of the implicit variables are less than the CONVERGE limit, the system is considered balanced.

CONSTITUTIVE EQUATIONS

Constitutive equations describe the relationships between the input variables, state variables, parameters, and external input variables of a device. For many devices, the constitutive equations are used to define implicit variables that have a value of zero when the constitutive laws are satisfied.

DEVICE

A **Device** is a type of electrical or mechanical machinery that can be included in a network description. The device is described by a number of constitutive equations that relate input variables to state variables, parameters, and external input variables. Examples of devices include resistors, synchronous generators, reduction gears, turbines and switches.

ELEMENT

An **Element** is a specific example of a device that has its own name and parameter values associated with it. A resistor for example, would be a device while R2 which has a value of 10 K ohms would be an element.

EXTERNAL INPUT VARIABLE

An **External Input Variable** allows the operator to specify an input to an element externally from the network description. The external input variables are specified in a Queue that is specified in the Simulation section of the input file. External input variables can also be initialized in the Initialize section.

EXTERNAL OUTPUT VARIABLE

External Output Variables are variables generated by an element that the operator may chose to display during the simulation. Only external output variables, external input variables, and voltage subnode voltages can be displayed.

IMPLICIT VARIABLE

Implicit Variables are calculated by each element from the input variables, parameters, state variables, and external input variables. The implicit variables have a value of zero when the constitutive laws governing the element are satisfied.

INPUT VARIABLE

Input Variables are used by the device descriptions to generate the implicit variables. Within the device description, the input variables are implicitly defined. Values are assigned to input variables by the network description.

JACOBIAN MATRIX

A **Jacobian Matrix** contains the partial derivatives of the implicit variables with respect to the input variables. In SEPSIP each element generates its own Jacobian Matrix. From the elemental Jacobian Matrices, a system Jacobina Matrix is generated that contains the partial derivatives of all the implicit variables with respect to all of the system variables.

KEYWORD

A **Keyword** is a word that is used by SEPSIP to delimit sections of the input file. Keywords are case insensitive and should not be used as Element names or Node names.

KIRCHHOFF's LAWS

Kirchhoff's Voltage and Current Laws determine the relation of currents and voltages at an electrical node. For a number of elements that are attached to a node, the sum of all the currents entering the node must equal zero. Furthermore, the voltage at a node is the same for each of the elements.

NETWORK

A **Network** specifies the interconnection of elements within a simulation. In SEPSIP, the Network is defined by assigning input variables to one of four types of subnodes that are in turn grouped into nodes.

NODAL EQUATIONS

Nodal Equations are the mathematical representations of Kirchhoff's Laws. The nodal equations are used to relate the input variables of all the elements of the system to the system variables.

NODE

A **Node** in electrical terms is a connection between terminals of two or more electrical elements. Associated with each node are two expressions which are mathematical statements of Kirchhoff's voltage and current laws. In SEPSIP, these mathematical relations are assigned to **subnodes**. A SEPSIP Node is used to group subnodes into easily understood groups. An Electrical Node is represented in SEPSIP by a SEPSIP Node having a voltage subnode and a current subnode.

PARAMETER

A **Parameter** is used to define an **element** as a specific example of a **device**. Parameters are assigned values in the first section of the SEPSIP input file and can not be changed during the simulation.

STATE VARIABLE

State Variables are variables internal to an element whose value at the end of the last time increment are saved and available. State variables are normally used for integration and differentiation. For elements that are described by state machines (such as circuit breakers), state variable can also be used to indicate the present state of the element.

SUBNODE

A **subnode** is used in SEPSIP to define the relation between the element input variables attached to it. There are four types of subnodes each of which specify a different relation for the attached variables.

SYSTEM JACOBIAN MATRIX

The **system Jacobian matrix** contains the partial derivatives of the implicit variables with respect to the system variables. This matrix is used to create corrections to the sytem variables which in turn are used to generate input variables that satisfy the constitutive relations of each element.

SYSTEM VARIABLE

The **system variables** are a set of independent variables from which all of the input variables of the elements can be derived from. The number of system variables must equal the total number of implicit variables. The system variables are defined by the four types of subnodes.

APPENDIX B

INSTRUCTIONS FOR ADDING DEVICES

Adding a device to SEPSIP is a five part process. First, a device driver routine must be written in the C programming language that calculates the implicit and external output variables along with the Jacobian matrix. The second step is to modify a special input file which SEPSIP uses to assign the number and variable names for each type of variable (state, input, etc.). The third step modifies an include file which notifies SEPSIP of the existence of the device and the name of the routine written in the first step. The fourth step involves adding the device driver file name to the **Makefile**. The Makefile assists in the recompilation procedure accomplished in the final step.

B.1 Write Device Driver Routines

The major role of the device driver routine is to calculate implicit variables based on the values of input, external input, state, old_state, and parameter variables in addition to the current simulation time increment. The routine can also calculate external output variables and a Jacobian matrix. The device driver routine may change the values of the implicit, external output, and state variables. Parameter, input, old_state, and external input variables should not normally be changed by device driver routines.

B.1.1 Arguments

A device driver routine should have the format shown in Figure B.1-1

Figure B.1-1 Device Driver Routine

```
/* f_dev_a.c */
#include <stdio.h>
#include <math.h>
#include "doerry.h"

/* The variables should be defined here using the define directive */

#define v0 e->con.in[0]
#define v1 e->con.in[1]
#define i0 e->con.in[2]
#define i1 e->con.in[3]

#define va     e->con.state[0]
#define va_old e->con.old_state[0]

#define R  e->con.param[0]
```

```
#define I1 e->con.implicit[0]

#define in_a  e->con.ext_in[0]
#define out_a e->con.ext_out[0]

dev_a(e,dt)
ELEMENT *e;
double dt;
{
   /* the code should be located here */
}
```

The **define** directives are optional but their use greatly eases the readability of the program. The **ELEMENT** structure along with the **CONNECT** structure are defined in **doerry.h**.

Figure B.1-2 ELEMENT and CONNECT structures

```
/* extracted from doerry.h */

/* These are the external input and external output types used
   in the type_ext_in and type_ext_out arrays of the Connect
   structure.  These types are presently unimplemented in SEPSIP
*/
#define BOOLEAN 0
#define SWITCH  1
#define INTEGER 2
#define FLOAT   3

/* The CONNECT structure holds arrays of variables associated
   with each element */

typedef struct Connect
{
  int nbr_inputs;          /* Number of input variables               */
  int nbr_states;          /* Number of internal states               */
  int nbr_implicit:        /* Number of implicit equations            */
  int nbr_ext_in;          /* Number of external input variables      */
  int nbr_ext_out;         /* Number of external output variables     */
  int nbr_param;           /* Number of parameters                    */
  double *in;              /* pointer to array of input variables     */
  double *state;           /* pointer to array of state variables     */
  double *old_state;       /* pointer to array of old state variables */
  double *implicit;        /* pointer to array of implicit variables  */
  double *ext_in;          /* pointer to array of external input
                              variables */
  double *ext_out;         /* pointer to array of external output
                              variables */
  double *param;           /* pointer to array of parameters          */
  double *init_state;      /* pointer to array of initial values for
                              state variables */
  double *init_ext_in;     /* pointer to array of initial values for
                              external input variables */
  double *init_in;         /* pointer to array of initial values for
                              input variables */
  double *jacob_in;        /* pointer to Jacobian matrix of implicit
                              variables with respect to input
                              variables */
  int jacob_switch;        /* = 1 if Jacobian calculated by function
                              = 0 if Jacobian calculated externally    */
  int *type_ext_in;        /* pointer to array of external input types */
  int *type_ext_out;       /* pointer to array of external output types*/
  int *imp_index;          /* pointer to array of indexes for itab
                              array */
}
CONNECT;
```

```
typedef struct Element
{
  int serial;             /* serial number of element (unused)      */
  char *name;             /* pointer to name of element             */
  struct Connect con;     /* Connect structure                      */
  struct Device *device;  /* Pointer to DEVICE structure which contains
                             the names of all the variables along with
                             the starting address of the device routine
                          */
  int flag;               /* = 1 if element is used in the network
                             = 0 if element is not used in the network */
}
ELEMENT;
```

B.1.2 Select number and types of variables

One of the first choices one must make when writing a device driver is the number of each type of variable. SEPSIP allocates only enough memory for each array to hold the number of variables specified in the Device Input File (see section B.2). The size of the allocated arrays are specified by the **nbr_*** variables in the **CONNECT** structure. Changing the **nbr_*** values will not reallocate the arrays and therefore should not be done.

Using the **define** directive as discussed in the previous section is a good way of assigning understandable labels to the memebers of the variable arrays. It is also a good idea to use the same name with both the **define** statement and the Device Input File (Section B.2).

B.1.3 Calculate Implicit variables

The heart of the device driver routine is the calculation of the implict variables. The calculations can involve any of the variables contained in the **ELEMENT** structure and the time incremnt **dt**. Generally, one should always check and write appropriate code for divide by zero situations.

B.1.4 Calculate State Variables

The state variables that are specified in the current time step are moved to the old_state variable array once the system is balanced. In this manner, one can store information that will be required in the following time increment.

B.1.5 Calculate External Output Variables (optional)

External Output variables allow the generation of variables that are not directly used in calculations but may be of interest to the user in monitoring the simulation. The original concept for SEPSIP was to be able to specify the external output variable to be one of four types, but presently only the **FLOAT** type is implemented. The typing refers only to the manner in which the variables are displayed and not to the way in which they are stored.

B.1.6 Calculate Jacobian Matrix (optional)

Calculating the Jacobian matrix within the device driver routine can greatly increase the speed of the simulations. The variable **e->con.jacob_switch** is used to indicate whether or not the Jacobian is calculated. if **e->con.jacob_switch** is zero, the Jacobian is not calculated by the routine. Otherwise, the Jacobian is calculated by the driver routine.

Each element of the Jacobian array is the partial derivative of an implicit variable with respect to one of the input variables. **e->con.jacob_in[i + N * j]** refers to the partial of **e->con.implicit[i]** with respect to **e->con.in[j]** where **N = e->con.nbr_implicit** is the total number of implicit variables.

B.2 Modify Device Input File (three_phase.input)

SEPSIP uses in input file to determine how many variables of each type should be allocated and the names of those variables. Each device must have an entry of the form shown in figure B.2-1. Appendix C includes two examples of input files.

Figure B.2-1 Device Input File Entry

```
NAME device_name
INPUTS 3
  input_name_1
  input_name_2
  input_name_3
STATES 2
  state_name_1
  state_name_2
IMPLICIT 2
  implicit_name_1
  implicit_name_2
EXTERNAL IN 3
  float   ext_in_1
  switch  ext_in_2
  integer ext_in_3
EXTERNAL OUT 2
  float   ext_out_1
```

```
  float   ext_out_2
PARAMETERS 2
  param_name_1
  param_name_2
END
```

The number following the variable type indicates how many variables of that type are listed.

For the external input and external output variables, a type indicator must be specified before the name of the variable. Legal types are **BOOLEAN, SWITCH, INTEGER,** and **FLOAT**. In the present version of SEPSIP, this indicator is ignored.

The variable names should all be unique within a device description. This means that one should not give an implicit variable the same name as an input variable. Some reuse of variable names is possible, but doing so can lead to much confusion.

B.3 Modify penner.h

The **penner.h** include file has two purposes. The first is to specify the names of the device input files and the number of devices described in these files. The second purpose is to associate device names with the device driver routines. Appendix C contains the current version of **penner.h**. Figure B.3-1 shows a simplified version of penner.h

Figure B.3-1 penner.h example

```
/* penner.h */

typedef int (*FUNCTION_PTR) ();

#define NBR_DEV_FILES 2  /* number of device input files */

        /* specify the names and paths of the device input files */

static char *device_file[] =
{
  "/mit/13.411/sepsip/three_phase.input",
  "/mit/13.411/sepsip/one_phase.input"
};

       /* specify the number of devices described in each of the
          above files */

static int nbr_device_file[] =
{
  3,
  1
};

       /* specify the names of the devices as declared in the
           the device input files */
```

```
static char *device_name[] =
{
  "t_line_3p",
  "rl_wye",
  "gen_synch_3p",
  "switch"
}

       /* specify the device driver routine corresponding to each of
         the above named devices */

#define F0 t_line_3p
#define F1 rl_wye
#define F2 gen_synch_3p
#define F3 spst_switch

       /* declare functions */

int F0 ();
int F1 ();
int F2 ();
int F3 ();

      /* declare function pointer array */

static FUNCTION_PTR dev_fnctn[] =
{
  F0,
  F1,
  F2,
  F3
}
```

The order of the device names in the **device_name** array should be the same as the order of the corresponding device driver routines in the **dev_fnctn** array.

B.4 Modify Makefile

Most computer systems include a **Make** utility for assisting in the management of software projects. The Make utiltity operates on a data file which in UNIX is usually named **Makefile**. The Makefile contains instructions as to which files should be compiled and linked in order to create an executable program. The Make utility uses these instructions to compile only those files that have changed since the last compilation. If only one file is edited, only that one file is recompiled and linked to the other object files. For this reason, **Make** can considerably reduce the compilation time of a large program. Figure B.4-1 shows an example of a Makefile.

Figure B.4-1 UNIX Makefile Example

```
FUNCO= f_t_line_3p.o f_rl_wye.o f_gen_synch_3p.o f_spst_switch.o
OBJB= check_name.o commands.o dump_data.o edit_simulate.o elm_jacob.o  \
      file_options.o gauss_eliminate.o integ.o ioliba.o load_device.o \
      load_element.o load_initial.o load_network.o load_simulation.o  \
      make_jacobian.o print_network.o read_device.o read_element.o    \
      read_network.o sepsip.o setup_simulation.o simulate.o           \
      dump_data.o
INCLUDEFILES = penner.h doerry.h
sepsip: $(OBJB) $(FUNCO)
        cc -c sepsip $(OBJB) $(FUNCO) -lm
load_device.o:  $(INCLUDEFILES)
        cc -c load_device.c
read_device.o:  $(INCLUDEFILES)
        cc -c read_device.c
sepsip.o:  $(INCLUDEFILES)
        cc -c sepsip.c
clobber:
        rm *.o
```

B.5 Recompile SEPSIP

The final step to add a device to SEPSIP is to recompile the program. Using the above Makefile, the recompilation is accomplished in UNIX by entering the command **make -k** at a UNIX prompt. For other systems, one must read the instruction manual for the system C compiler.

APPENDIX C

DEVICE DRIVER CODE

C.1 f_t_line_3p.c

f_t_line_3p.c

```
/* f_t_line_3p.c */
/* Norbert H. Doerry
   11 March 1989

   This routine simulates a 3 phase transmission line as a series
   combination of a resistance and reactance.  The reactance also
   has a parallel leakage resistance
*/
#include <stdio.h>
#include <math.h>
#include "doerry.h"


#define v0a e->con.in[0]
#define v0b e->con.in[1]
#define v0c e->con.in[2]
#define v1a e->con.in[3]
#define v1b e->con.in[4]
#define v1c e->con.in[5]
#define i0a e->con.in[6]
#define i0b e->con.in[7]
#define i0c e->con.in[8]
#define i1a e->con.in[9]
#define i1b e->con.in[10]
#define i1c e->con.in[11]
#define va  e->con.state[0]
#define vb  e->con.state[1]
#define vc  e->con.state[2]
#define ia  e->con.state[3]
#define ib  e->con.state[4]
#define ic  e->con.state[5]
#define va_old  e->con.old_state[0]
#define vb_old  e->con.old_state[1]
#define vc_old  e->con.old_state[2]
#define ia_old  e->con.old_state[3]
#define ib_old  e->con.old_state[4]
#define ic_old  e->con.old_state[5]
#define R e->con.param[0]
#define L e->con.param[1]
#define R1 e->con.param[2]
#define v0a_ 0
#define v0b_ 1
#define v0c_ 2
#define v1a_ 3
#define v1b_ 4
#define v1c_ 5
#define i0a_ 6
#define i0b_ 7
#define i0c_ 8
#define i1a_ 9
#define i1b_ 10
#define i1c_ 11

t_line_3p(e,dt)
```

```
ELEMENT *e;
double dt;
{
  double fta,ftb,ftc,fta_old,ftb_old,ftc_old;
  double ila,ilb,ilc,ila_old,ilb_old,ilc_old;
  int i,j;

  /* initialize the jacobian matrix to zeroes if dt  == 0 */

  for (i = 0 ; dt == 0.0  && i < 6 ; i++)
    for (j = 0 ; j < 12 ; j++)
      e->con.jacob_in[i + 6 * j] = 0.0;



  va = v1a - v0a;
  vb = v1b - v0b;
  vc = v1c - v0c;

  ia = (i1a - i0a) / 2.0;
  ib = (i1b - i0b) / 2.0;
  ic = (i1c - i0c) / 2.0;


  /* see if the inductance is zero or the leakage resistance is zero*/

  if (L == 0 || R1 == 0)
    {
      /* pure resistance */

      e->con.implicit[0] = va - ia * R;
      e->con.implicit[1] = vb - ib * R;
      e->con.implicit[2] = vc - ic * R;

      e->con.jacob_in[0 + 6 * v0a_] =  -1.0;
      e->con.jacob_in[0 + 6 * v1a_] =   1.0;
      e->con.jacob_in[0 + 6 * i0a_] =   R / 2.0;
      e->con.jacob_in[0 + 6 * i1a_] =  -R / 2.0;

      e->con.jacob_in[1 + 6 * v0b_] =  -1.0;
      e->con.jacob_in[1 + 6 * v1b_] =   1.0;
      e->con.jacob_in[1 + 6 * i0b_] =   R / 2.0;
      e->con.jacob_in[1 + 6 * i1b_] =  -R / 2.0;

      e->con.jacob_in[2 + 6 * v0c_] =  -1.0;
      e->con.jacob_in[2 + 6 * v1c_] =   1.0;
      e->con.jacob_in[2 + 6 * i0c_] =   R / 2.0;
      e->con.jacob_in[2 + 6 * i1c_] =  -R / 2.0;
    }

  else
    {
      /* inductance present */
```

```
    /* inductor voltage divided by inductance */

    fta = (va - R * ia) / L;
    ftb = (vb - R * ib) / L;
    ftc = (vc - R * ic) / L;

    /* find inductor current */

    ila = ia - fta * L / R1;
    ilb = ib - ftb * L / R1;
    ilc = ic - ftc * L / R1;

    /* find old values */

    fta_old = (va_old - R * ia_old) / L;
    ftb_old = (vb_old - R * ib_old) / L;
    ftc_old = (vc_old - R * ic_old) / L;

    ila_old = ia_old - fta_old * L / R1;
    ilb_old = ib_old - ftb_old * L / R1;
    ilc_old = ic_old - ftc_old * L / R1;

    /* calculate implicit variables using trapezoidal integeration */

    e->con.implicit[0] = integ(ila,ila_old,fta,fta_old,dt);
    e->con.implicit[1] = integ(ilb,ilb_old,ftb,ftb_old,dt);
    e->con.implicit[2] = integ(ilc,ilc_old,ftc,ftc_old,dt);

    /* calculate Jacobian Matrix */

    e->con.jacob_in[0 + 6 * v0a_] = - 1.0 / R1 - dt / (2.0 * L);
    e->con.jacob_in[0 + 6 * v1a_] = - e->con.jacob_in[0 + 6 * v0a_];
    e->con.jacob_in[0 + 6 * i0a_] = - (1.0 + R / R1 +
                                       (dt * R) / (2.0 * L)) / 2.0;
    e->con.jacob_in[0 + 6 * i1a_] = - e->con.jacob_in[0 + 6 * v0a_];

    e->con.jacob_in[1 + 6 * v0b_] = - 1.0 / R1 - dt / (2.0 * L);
    e->con.jacob_in[1 + 6 * v1b_] = - e->con.jacob_in[1 + 6 * v0b_];
    e->con.jacob_in[1 + 6 * i0b_] = - (1.0 + R / R1 +
                                       (dt * R) / (2.0 * L)) / 2.0;
    e->con.jacob_in[1 + 6 * i1b_] = - e->con.jacob_in[1 + 6 * v0b_];

    e->con.jacob_in[2 + 6 * v0c_] = - 1.0 / R1 - dt / (2.0 * L);
    e->con.jacob_in[2 + 6 * v1c_] = - e->con.jacob_in[2 + 6 * v0c_];
    e->con.jacob_in[2 + 6 * i0c_] = - (1.0 + R / R1 +
                                       (dt * R) / (2.0 * L)) / 2.0;
    e->con.jacob_in[2 + 6 * i1c_] = - e->con.jacob_in[2 + 6 * v0c_];
  }

/* current sums are the following three implicit equations */

e->con.implicit[3] = i0a + i1a;
e->con.implicit[4] = i0b + i1b;
e->con.implicit[5] =.i0c + i1c;
```

```
  e->con.jacob_in[3 + 6 * i0a_] = 1.0;
  e->con.jacob_in[3 + 6 * i1a_] = 1.0;
  e->con.jacob_in[4 + 6 * i0b_] = 1.0;
  e->con.jacob_in[4 + 6 * i1b_] = 1.0;
  e->con.jacob_in[5 + 6 * i0c_] = 1.0;
  e->con.jacob_in[5 + 6 * i1c_] = 1.0;


  /* turn the jacob switch on */

  e->con.jacob_switch = 1;

  /* store external output variables */

  for ( i = 0 ; i < 6 ; i++)
    e->con.ext_out[i] = e->con.state[i];



}
```

C.2 f_rl_wye.c

```
/* f_rl_wye.c */
/* Norbert H Doerry

   12 March 1898

   This file simulates a three phase rl load connected in a wye fashion.

   The center point, or ground, can be connected to a reference current
   subnode if it is desired to leave the line floating.  The neutral
   voltage can alternately be set to a specific value by using a reference
   voltage subnode.

*/
#include <stdio.h>
#include <math.h>
#include "doerry.h"

#define v0a e->con.in[0]
#define v0b e->con.in[1]
#define v0c e->con.in[2]
#define v0n e->con.in[3]
#define i0a e->con.in[4]
#define i0b e->con.in[5]
#define i0c e->con.in[6]
#define i0n e->con.in[7]
#define va  e->con.state[0]
#define vb  e->con.state[1]
#define vc  e->con.state[2]
#define ia  e->con.state[3]
#define ib  e->con.state[4]
#define ic  e->con.state[5]
#define va_old  e->con.old_state[0]
#define vb_old  e->con.old_state[1]
#define vc_old  e->con.old_state[2]
#define ia_old  e->con.old_state[3]
#define ib_old  e->con.old_state[4]
#define ic_old  e->con.old_state[5]
#define v0a_ 0
#define v0b_ 1
#define v0c_ 2
#define v0n_ 3
#define i0a_ 4
#define i0b_ 5
#define i0c_ 6
#define i0n_ 7
#define R  e->con.param[0]
#define L  e->con.param[1]
#define R1 e->con.param[2]

rl_wye(e,dt)
ELEMENT *e;
double dt;
{
  int i,j;
```

```
  double fta,ftb,ftc,fta_old,ftb_old,ftc_old;
  double i1a,i1b,i1c;
  double Re;

  /* initialize the jacobian matrix to zeroes */

  for (i = 0 ; dt == 0.0 && i < 4 ; i++)
    for (j = 0 ; j < 8 ; j++)
      e->con.jacob_in[i + 4 * j] = 0.0;

  /* calculate states */

  i1a = i0n + i0b + i0c;
  i1b = i0n + i0c + i0a;
  i1c = i0n + i0a + i0b;

  va = v0a - v0n;
  vb = v0b - v0n;
  vc = v0c - v0n;
  ia = (i0a - i1a) / 2.0;
  ib = (i0b - i1b) / 2.0;
  ic = (i0c - i1c) / 2.0;

  /* sum of currents should be zero */

  e->con.implicit[3] = i0a + i0b + i0c + i0n;

  e->con.jacob_in[3 + 4 * i0a_] = 1.0;
  e->con.jacob_in[3 + 4 * i0b_] = 1.0;
  e->con.jacob_in[3 + 4 * i0c_] = 1.0;
  e->con.jacob_in[3 + 4 * i0n_] = 1.0;

  /* see if inductance is neglible */

  if (L == 0 || R1 == 0)
    {
      /* pure resistance */

      Re = (R1 == 0 || R == 0) ? 0.0 : R1 * R / (R1 + R);

      e->con.implicit[0] = va - ia * Re;
      e->con.implicit[1] = vb - ib * Re;
      e->con.implicit[2] = vc - ic * Re;

      if (dt == 0)
        {

          e->con.jacob_in[0 + 4 * v0a_] =   1.0;
          e->con.jacob_in[0 + 4 * v0n_] =  -1.0;
          e->con.jacob_in[0 + 4 * i0a_] =  -Re / 2.0;
          e->con.jacob_in[0 + 4 * i0b_] =   Re / 2.0;
          e->con.jacob_in[0 + 4 * i0c_] =   Re / 2.0;
          e->con.jacob_in[0 + 4 * i0n_] =   Re / 2.0;
```

```
          e->con.jacob_in[1 + 4 * v0b_] =   1.0;
          e->con.jacob_in[1 + 4 * v0n_] =  -1.0;
          e->con.jacob_in[1 + 4 * i0b_] =  -Re / 2.0;
          e->con.jacob_in[1 + 4 * i0c_] =   Re / 2.0;
          e->con.jacob_in[1 + 4 * i0a_] =   Re / 2.0;
          e->con.jacob_in[1 + 4 * i0n_] =   Re / 2.0;

          e->con.jacob_in[2 + 4 * v0c_] =   1.0;
          e->con.jacob_in[2 + 4 * v0n_] =  -1.0;
          e->con.jacob_in[2 + 4 * i0c_] =  -Re / 2.0;
          e->con.jacob_in[2 + 4 * i0a_] =   Re / 2.0;
          e->con.jacob_in[2 + 4 * i0b_] =   Re / 2.0;
          e->con.jacob_in[2 + 4 * i0n_] =   Re / 2.0;
        }
    }

  else
    {
      /* inductance present */

      e->con.implicit[0] = ia - ia_old - (dt / L) *
        ((va + va_old) / 2.0 - R * ia_old) - (va - va_old) / R1;
      e->con.implicit[1] = ib - ib_old - (dt / L) *
        ((vb + vb_old) / 2.0 - R * ib_old) - (vb - vb_old) / R1;
      e->con.implicit[2] = ic - ic_old - (dt / L) *
        ((vc + vc_old) / 2.0 - R * ic_old) - (vc - vc_old) / R1;

      e->con.jacob_in[0 + 4 * v0a_] = - dt / (2.0 * L) - 1.0 / R1;
      e->con.jacob_in[0 + 4 * v0n_] = - e->con.jacob_in[0 + 4 * v0a_];
      e->con.jacob_in[0 + 4 * i0a_] = 0.5;
      e->con.jacob_in[0 + 4 * i0b_] = - e->con.jacob_in[0 + 4 * i0a_];
      e->con.jacob_in[0 + 4 * i0c_] = - e->con.jacob_in[0 + 4 * i0a_];
      e->con.jacob_in[0 + 4 * i0n_] = - e->con.jacob_in[0 + 4 * i0a_];

      e->con.jacob_in[1 + 4 * v0b_] = - dt / (2.0 * L) - 1.0 / R1;
      e->con.jacob_in[1 + 4 * v0n_] = - e->con.jacob_in[1 + 4 * v0b_];
      e->con.jacob_in[1 + 4 * i0b_] = 0.5;
      e->con.jacob_in[1 + 4 * i0c_] = - e->con.jacob_in[1 + 4 * i0b_];
      e->con.jacob_in[1 + 4 * i0a_] = - e->con.jacob_in[1 + 4 * i0b_];
      e->con.jacob_in[1 + 4 * i0n_] = - e->con.jacob_in[1 + 4 * i0b_];

      e->con.jacob_in[2 + 4 * v0c_] = - dt / (2.0 * L) - 1.0 / R1;
      e->con.jacob_in[2 + 4 * v0n_] = - e->con.jacob_in[2 + 4 * v0c_];
      e->con.jacob_in[2 + 4 * i0c_] = 0.5;
      e->con.jacob_in[2 + 4 * i0a_] = - e->con.jacob_in[2 + 4 * i0c_];
      e->con.jacob_in[2 + 4 * i0b_] = - e->con.jacob_in[2 + 4 * i0c_];
      e->con.jacob_in[2 + 4 * i0n_] = - e->con.jacob_in[2 + 4 * i0c_];

    }

  /* turn the jacob switch on */

  e->con.jacob_switch = 1;
/*  e->con.jacob_switch = 0; */
```

```
  /* store external output variables */

  for ( i = 0 ; i < 6 ; i++)
    e->con.ext_out[i] = e->con.state[i];

}
```

C.3 f_synch_mach.c

```
/* f_synch_mach.c */
/* Norbert H. Doerry

   20 April 1989

   This file contains the driver routine for simulating a generic
   synchronous machine as modelled in:

            Synchronous Machine Dynamic Models

            J. L. Kirtley Jr.
            LEES Technical Report TR-87-008

            June 5, 1987.


   ****  Modified 29 april ******

   Changed to use modified trapezoidal integration  -nhd

*/

#include <stdio.h>
#include <math.h>
#include "doerry.h"

/* terminal voltages and currents */

#define v0a e->con.in[0]
#define v0b e->con.in[1]
#define v0c e->con.in[2]
#define v0n e->con.in[3]
#define i0a e->con.in[4]
#define i0b e->con.in[5]
#define i0c e->con.in[6]

/* field voltages and currents */

#define v0f e->con.in[7]
#define v1f e->con.in[8]
#define i0f e->con.in[9]
#define i1f e->con.in[10]

/* rotational properties */
/* Theta is electrical radians,  wm and wm_dot are mechanical */

#define theta e->con.in[11]
#define wm    e->con.in[12]
#define wm_dt e->con.in[13]

/* sum of rotational inertia and electrical torque */
/* Loads are a negative Te,  prime movers provide a positive Te */
#define Te    e->con.in[14]
```

```
/* internal variable */

#define Psi_q e->con.in[15]
#define Psi_d e->con.in[16]


/* define offsets */

#define v0a_       0
#define v0b_       1
#define v0c_       2
#define v0n_       3
#define i0a_       4
#define i0b_       5
#define i0c_       6
#define v0f_       7
#define v1f_       8
#define i0f_       9
#define i1f_      10
#define Theta_    11
#define wm_       12
#define wm_dt_    13
#define Te_       14

/* define parameters */

#define xd      e->con.param[0]
#define xq      e->con.param[1]
#define xd_p    e->con.param[2]
#define xd_pp   e->con.param[3]
#define xq_pp   e->con.param[4]
#define xal     e->con.param[5]
#define Tdo_p   e->con.param[6]
#define Tdo_pp  e->con.param[7]
#define Tqo_pp  e->con.param[8]
#define Tad     e->con.param[9]
#define Ifnl    e->con.param[10]
#define H       e->con.param[11]
#define pp      e->con.param[12]
#define wbs     e->con.param[13]
#define Vdb     e->con.param[14]
#define Pbs     e->con.param[15]


/* define states */

#define s_theta    e->con.state[0]
#define s_wm       e->con.state[1]
#define s_wm_dt    e->con.state[2]
#define psi_d      e->con.state[3]
#define psi_q      e->con.state[4]
#define eq_p       e->con.state[5]
#define eq_pp      e->con.state[6]
#define ed_pp      e->con.state[7]
```

```
#define d_psi_d   e->con.state[8]
#define d_psi_q   e->con.state[9]
#define d_eq_p    e->con.state[10]
#define d_eq_pp   e->con.state[11]
#define d_ed_pp   e->con.state[12]


#define s_theta_old   e->con.old_state[0]
#define s_wm_old      e->con.old_state[1]
#define s_wm_dt_old   e->con.old_state[2]
#define psi_d_old     e->con.old_state[3]
#define psi_q_old     e->con.old_state[4]
#define eq_p_old      e->con.old_state[5]
#define eq_pp_old     e->con.old_state[6]
#define ed_pp_old     e->con.old_state[7]
#define d_psi_d_old   e->con.old_state[8]
#define d_psi_q_old   e->con.old_state[9]
#define d_eq_p_old    e->con.old_state[10]
#define d_eq_pp_old   e->con.old_state[11]
#define d_ed_pp_old   e->con.old_state[12]

/* define implicit variables */

#define isum      e->con.implicit[0]
#define ifsum     e->con.implicit[1]
#define vo        e->con.implicit[2]
#define i_psi_d   e->con.implicit[3]
#define i_psi_q   e->con.implicit[4]
#define i_eq_pp   e->con.implicit[5]
#define i_ed_pp   e->con.implicit[6]
#define i_eq_p    e->con.implicit[7]
#define Torq      e->con.implicit[8]
#define W         e->con.implicit[9]
#define Wdot      e->con.implicit[10]

#define PI2_3     2.0943951
#define C2_3      0.66666666667
#define TWOPI     6.2831853

#define xad       e->con.ext_out[0]
#define xkd       e->con.ext_out[1]
#define xf        e->con.ext_out[2]
#define rf        e->con.ext_out[3]

#define IdB       e->con.ext_out[4]
#define IfB       e->con.ext_out[5]
#define VfB       e->con.ext_out[6]
#define Tbs       e->con.ext_out[7]

#define alpha     e->con.ext_out[8]
#define ifd       e->con.ext_out[9]
#define vfd       e->con.ext_out[10]
#define eaf       e->con.ext_out[11]
#define vd        e->con.ext_out[12]
```

```
#define vq          e->con.ext_out[13]
#define id          e->con.ext_out[14]
#define iq          e->con.ext_out[15]

#define Tepu        e->con.ext_out[16]

#define RPM         e->con.ext_out[17]
#define Pmech       e->con.ext_out[18]
#define Pe          e->con.ext_out[19]
#define Tacc        e->con.ext_out[20]

#define Ia          e->con.ext_out[21]
#define Ib          e->con.ext_out[22]
#define Ic          e->con.ext_out[23]

synch_mach(e,dt)
ELEMENT *e;
double dt;
{
  double temp;
  double va,vb,vc;
  int i,j;
  double io;
  double cost,costp,costm,sint,sintp,sintm;
  double Taq;

  Ia = i0a;
  Ib = i0b;
  Ic = i0c;

  /* ensure theta is in range */

/*  while (theta >  TWOPI) theta -= TWOPI;
  while (theta < -TWOPI) theta += TWOPI; */


  cost  = cos(theta);
  costp = cos(theta + PI2_3);
  costm = cos(theta - PI2_3);
  sint  = sin(theta);
  sintp = sin(theta + PI2_3);
  sintm = sin(theta - PI2_3);

  /* calculate RPM */

  RPM = wm * 60.0 / TWOPI;


  /* using the shielding constraint model  */

  /* calculate base quantities */

  if (wbs == 0) wbs = TWOPI * 60.0;
  if (Vdb == 0) Vdb = 1.0;
```

```
if (Pbs == 0) Pbs = 1.0;
if (pp <=  0) pp  = 1.0;

IdB = Pbs * C2_3 / Vdb;
IfB = Ifnl * (xd - xal);

if (IfB == 0) IfB = 1.0;
VfB = Pbs / IfB;

Tbs = pp * Pbs / wbs;

/* calculate the phase voltages */

va = v0a - v0n;
vb = v0b - v0n;
vc = v0c - v0n;

/* calculate instantaneous power */

Pe = va * i0a + vb * i0b + vc * i0c;

/* do the Parks transformation for both the current and voltages */

id =   C2_3 * (cost  * i0a + costm * i0b + costp * i0c) / IdB;
iq = - C2_3 * (sint  * i0a + sintm * i0b + sintp * i0c) / IdB;
io =   C2_3 * ( i0a / 2.0  +  i0b / 2.0  +  i0c / 2.0 ) / IdB;

vd =   C2_3 * (cost  * va + costm * vb + costp * vc) / Vdb;
vq = - C2_3 * (sint  * va + sintm * vb + sintp * vc) / Vdb;
vo =   C2_3 * ( va / 2.0  +  vb / 2.0  +  vc / 2.0 ) / Vdb;

/* convert the field variables to per unit */

ifd = (i1f - i0f) / (2.0 * IfB);
vfd = (v1f - v0f) / VfB;

Taq = (xd_pp != 0.0) ? Tad * xq_pp / xd_pp : Tad;

/* ensure xd_pp and xq_pp are non zero */

if (xd_pp <= 0.0) xd_pp = .01;
if (xq_pp <= 0.0) xq_pp = .01;

/* calculate parameters */

xf  = (xd != xd_p)   ? (xd - xal) * (xd - xal) / (xd - xd_p) : 1000.0;
rf  = (Tdo_p != 0.0) ?  xf / (wbs * Tdo_p) : 1000.0;
xad = xd - xal;
xkd = (xd != xd_pp)  ? xad * xad / (xd - xd_pp) : 1000.0;
alpha = (xd_p != xd_pp) ? (xd - xd_pp) / (xd_p - xd_pp) : 1000.0;

/* calculate variables */

eaf = (rf != 0.0)    ? xad * vfd / rf  : 1000.0;
```

```
ed_pp = xq_pp * iq - Psi_q;
eq_pp = Psi_d - xd_pp * id;
eq_p = (xf != 0.0)   ? (xad * (xf - xkd) * ifd + xkd * eq_pp) / xf :
  xad * ifd;
psi_d = Psi_d;
psi_q = Psi_q;

/* calculate derivatives */

d_psi_d = (Tad != 0) ? ( eq_pp - psi_d) / Tad + wm * pp * psi_q + wbs * vd :
  0.0;
d_psi_q = (Taq != 0) ? (-psi_q - ed_pp) / Taq - wm * pp * psi_d + wbs * vq :
  0.0;
d_eq_pp = (Tdo_pp != 0) ? (- xd_p * eq_pp / xd_pp +
          eq_p + (xd_p - xd_pp) * psi_d / xd_pp) / Tdo_pp : 0.0;
d_ed_pp = (Tqo_pp != 0) ? (- xq * ed_pp / xq_pp -
                           (xq - xq_pp) * psi_q / xq_pp) / Tqo_pp : 0.0;
d_eq_p  = (Tdo_p != 0) ?
  (-alpha * eq_p + (alpha - 1.0) * eq_pp + eaf) / Tdo_p : 0.0;

/* calculate associated implicit variables (trapezoidal integeration) */
/* changed 29 april to use modified trapezoidal method  */

i_psi_d = (Tad != 0) ?
  psi_d - psi_d_old - (dt) * (0.6 * d_psi_d + 0.4 * d_psi_d_old) :
    eq_pp - psi_d;
i_psi_q = (Taq != 0) ?
  psi_q - psi_q_old - (dt) * (0.6 * d_psi_q + 0.4 * d_psi_q_old) :
    psi_q + ed_pp;
i_eq_pp = (Tdo_pp != 0) ?
  eq_pp - eq_pp_old - (dt) * (0.6 * d_eq_pp + 0.4 * d_eq_pp_old) :
    - xd_p * eq_pp / xd_pp + eq_p + (xd_p - xd_pp) * psi_d / xd_pp ;
i_ed_pp = (Tqo_pp != 0) ?
  ed_pp - ed_pp_old - (dt) * (0.6 * d_ed_pp + 0.4 * d_ed_pp_old) :
    - xq * ed_pp / xq_pp - (xq - xq_pp) * psi_q / xq_pp;
i_eq_p  = (Tdo_p != 0) ?
  eq_p  - eq_p_old  - (dt) * (0.6 * d_eq_p  + 0.4 * d_eq_p_old ) :
    -alpha * eq_p + (alpha - 1.0) * eq_pp + eaf;

/* the zero sequence current should go to zero since sum of currents
 must go to zero if there is no leakage to ground */

isum = io;

/* the sum of the field currents should be zero */

ifsum = (i0f + i1f) / IfB;

/* vo also goes to zero since the zero sequence current is always zero
   and therefore the zero sequence flux can never build up */


/* calculate per unit torque of electrical origin */
```

```
  Tepu = psi_d * iq - psi_q * id;   /* 59 */

  /* calculate the mechanical Power */

  Pmech = Tepu * wm * Tbs;

  /* calculate load torque */

  Tacc  =  2.0 * H * pp * wm_dt / wbs;

  /* The Te variable is Nm of the torque coming out of the machine */

  Torq = Tacc - Tepu - Te / Tbs;

  /* integrate the frequency */

  s_theta = theta;
  s_wm = wm;
  s_wm_dt = wm_dt;

  W = s_theta - s_theta_old - (dt / 2.0) * (s_wm * pp + s_wm_old * pp);

  /* modified the trapezoidal integration to weight the 'Euler Backward'
     contribution.  This prevents the acceleration from oscillating
     should the frequency be held at a constant
     */

  Wdot = s_wm - s_wm_old - (dt) * (0.6 * s_wm_dt + 0.4 * s_wm_dt_old);

  /* turn jacobian switch off */

  e->con.jacob_switch = 0.0;
}
```

C.4 f_speed_reg.c

```
/* f_speed_reg.c */

/* Norbert H. Doerry
   15 April 1989

   This device describes a prime mover controlled by a mechancial
   governor and attached to a shaft that has windage losses (B).

   The governor is based on the type found on the SSN-637 class
   submarine and described in:

   R.C. Dalton
   Turbine Generator Simulations for DD-692 Class 450 KW Machine
   and SSN-637 Class 2000 KW Machine, NAVSSES Philadelphia Project C-267.
   5 March 1984.

   This particular model was developed in:

   Norbert H. Doerry
   Shipboard Electrical Generator Simulation
   Semester Project Report for 6.238, MIT
   17 May 1988

   **** Modified   1 May 1989   ******

   changed the control from speed to torque

   *********



   Input variables

   s          = Primary Amplifier Fulcrum Displacement (inches) (0 to .5)
   Tm         = Torque on shaft (Nm)
   wm         = Mechanical speed (rad/sec)

   Parameters

   wnlo       = 374.72 (rad/sec)      base setting for speed
   wds        =  63.38 (rad/sec-in)   Coefficient for s
   wdTepu     = -20.15 (rad/sec)      Coefficient for Tmm / TBS
   TBS        = Base Torque
   Tg         = .3275 (sec)           Regulator time constant
   B          =       (Nmsec)         Damping Coeffient

   External Output variables

   Tmm        = Torque seen by prime mover
   Wg         = (HZ) Droop frequency
   Pshaft     = power seen at shaft
   Pdeliver   = power delivered by the prime mover
*/
```

f_speed_reg.c

```
#include <stdio.h>
#include <math.h>
#include "doerry.h"

#define wm              e->con.in[0]
#define Tm              e->con.in[1]
#define s               e->con.in[2]

#define Tm_order        e->con.state[0]
#define Tmpu            e->con.state[1]
#define Tm_order_old    e->con.old_state[0]
#define Tmpu_old        e->con.old_state[1]

#define wnlo            e->con.param[0]
#define wds             e->con.param[1]
#define wdTepu          e->con.param[2]
#define TBS             e->con.param[3]
#define Tg              e->con.param[4]
#define B               e->con.param[5]

#define Tmm             e->con.ext_out[0]
#define Tm_             e->con.ext_out[1]
#define Pshaft          e->con.ext_out[2]
#define Pdeliver        e->con.ext_out[3]

#define DEG_RAD 57.29578
#define RAD_HZ 6.28319
#define WBS  377.0

speed_reg(e,dt)
ELEMENT *e;
double dt;
{

  Tmm = Tm + B * wm;

  if (TBS == 0) TBS = 1.0;

  Tmpu = Tmm / TBS;

  if (wdTepu != 0)
    {

      /* find the desired torque */

      Tm_order = (wm - wnlo - wds * s) / wdTepu;

      e->con.implicit[0] = (Tmpu - Tmpu_old) * Tg -
        dt * (Tm_order_old - .5 * Tmpu - .5 * Tmpu_old);

      e->con.jacob_in[0] = (B   / TBS) * (Tg - dt * .5);
      e->con.jacob_in[1] = (1.0 / TBS) * (Tg - dt * .5);
      e->con.jacob_in[2] = 0.0;
```

```
    }
  else
    {
      /* set the speed to the ordered speed */

      e->con.implicit[0] = (wm - wnlo - wds * s) / WBS;

      e->con.jacob_in[0] = 1.0 / WBS;
      e->con.jacob_in[1] = 0.0;
      e->con.jacob_in[2] = - wds / WBS;

      Tm_order = Tmpu;
    }

  Tm_ = Tm_order;

  e->con.jacob_switch = 1.0;

  Pshaft = Tm * wm;
  Pdeliver = Tmm * wm;



}
```

C.5 f_volt_reg.c

f_volt_reg.c

```
/* f_volt_reg.c */

/* Norbert H. Doerry
   15 April 1989

*/

#include <stdio.h>
#include <math.h>
#include "doerry.h"

#define v0a     e->con.in[0]
#define v0b     e->con.in[1]
#define v0c     e->con.in[2]
#define v0f     e->con.in[3]
#define v1f     e->con.in[4]
#define i0f     e->con.in[5]
#define i1f     e->con.in[6]

#define Vbs     e->con.in[7]       /* This is the desired voltage */

#define wbs     e->con.in[8]
#define phase   e->con.in[9]
#define vt      e->con.in[10]

#define Vfdbs   e->con.param[0]     /* This is nominal field voltage    */
#define K       e->con.param[1]     /* This is forward DC gain of error */
#define Tvr     e->con.param[2]     /* This if voltage regulator Time const */
#define Vfmax   e->con.param[3]     /* maximum limit for field voltage  */
#define Vfmin   e->con.param[4]     /* minimum limit for field voltage  */

#define I1          e->con.implicit[0]
#define I2          e->con.implicit[1]
#define Isum        e->con.implicit[2]
#define Integrate   e->con.implicit[3]

#define Verr    e->con.state[0]
#define Vsig    e->con.state[1]
#define theta   e->con.state[2]
#define clip    e->con.state[3]

#define Verr_old e->con.old_state[0]
#define Vsig_old e->con.old_state[1]
#define theta_old e->con.old_state[2]
#define clip_old  e->con.old_state[3]

#define PI2_3        2.0943951
#define TWOPI        6.2831853


volt_reg(e,dt)
ELEMENT *e;
double dt;
{
```

```
  double va,vb,vc,vn,vsig,vf;
  double cost,costm;

  vn = (v0a + v0b + v0c) / 3.0;

  va = v0a - vn;
  vb = v0b - vn;
  vc = v0c - vn;

  if (vt < 0)
    {
      vt    *= -1.0;
      phase -= PI;
    }

  /* calculate phase */

  while (theta_old >  TWOPI) theta_old -= TWOPI;
  while (theta_old < -TWOPI) theta_old += TWOPI;

  /* keep the phase angle in a good range */

  while (phase >  TWOPI) phase -= TWOPI;
  while (phase < -TWOPI) phase += TWOPI;

  theta = theta_old + wbs * dt;   /* euler backwards method */

  cost  = cos(theta + phase);
  costm = cos(theta - PI2_3 + phase);

  I1 = (va - vt * cost ) / Vbs;
  I2 = (vb - vt * costm) / Vbs;


  Verr = 1.0 - vt / Vbs;

  Vsig = (v1f - v0f) / Vfdbs - 1.0;

  if (clip_old == 0)
    {

      /* use a modified trapezoidal integration scheme (weight euler back)*/

      Integrate = Tvr * (Vsig - Vsig_old) +
        dt * ( .6 * Vsig + .4 * Vsig_old - .6 * K * Verr - .4 * K * Verr_old);

      vf = v1f - v0f;

    }
  else if (clip_old == 1)
    {
      Integrate = (Vfmax - (v1f - v0f)) / Vbs;

      vsig = (Tvr * Vsig_old -
```

```
                dt * (.4 * Vsig_old - .6 * K * Verr - .4 * K * Verr_old)) /
                (Tvr + dt * 0.6);

        vf = (vsig + 1.0) * Vfdbs;

      }
    else
      {
        Integrate = (Vfmin - (v1f - v0f)) / Vbs;
        vsig = (Tvr * Vsig_old -
                dt * (.4 * Vsig_old - .6 * K * Verr - .4 * K * Verr_old)) /
                (Tvr + dt * 0.6);

        vf = (vsig + 1.0) * Vfdbs;
      }

    Isum = i0f + i1f;


    /* update the state */

    if (vf >= Vfmax) clip = 1.0;
    else if (vf <= Vfmin) clip = -1.0;
    else clip = 0.0;

    /* let the system calculate the jacobian for now */

    e->con.jacob_switch = 0.0;

}
```

C.6 f_ind_motor.c

f_ind_motor.c

```
/* f_ind_motor.c */

/* Norbert H. Doerry
   15 April 1989

*/

#include <stdio.h>
#include <math.h>
#include "doerry.h"

#define v0a     e->con.in[0]
#define v0b     e->con.in[1]
#define v0c     e->con.in[2]
#define i0a     e->con.in[3]
#define i0b     e->con.in[4]
#define i0c     e->con.in[5]
#define theta   e->con.in[6]
#define wm      e->con.in[7]
#define wm_dt   e->con.in[8]
#define v0n     e->con.in[9]
#define ira     e->con.in[10]
#define irb     e->con.in[11]
#define irc     e->con.in[12]


#define Ila     e->con.implicit[0]
#define Ilb     e->con.implicit[1]
#define Ilc     e->con.implicit[2]
#define Ilra    e->con.implicit[3]
#define Ilrb    e->con.implicit[4]
#define Ilrc    e->con.implicit[5]
#define Isum    e->con.implicit[6]
#define W       e->con.implicit[7]
#define Wdot    e->con.implicit[8]
#define Torque  e->con.implicit[9]


#define Rs              e->con.param[0]
#define Xls             e->con.param[1]
#define XM              e->con.param[2]
#define Xlr_prime       e->con.param[3]
#define Rr_prime        e->con.param[4]
#define J               e->con.param[5]
#define wbs             e->con.param[6]
#define pp              e->con.param[7]
#define B               e->con.param[8]

#define Tmech           e->con.ext_in[0]

#define lam_sa          e->con.state[0]
#define lam_sb          e->con.state[1]
#define lam_sc          e->con.state[2]
#define dlam_sa         e->con.state[3]
```

f_ind_motor.c

```
#define dlam_sb         e->con.state[4]
#define dlam_sc         e->con.state[5]
#define lam_ra_p        e->con.state[6]
#define lam_rb_p        e->con.state[7]
#define lam_rc_p        e->con.state[8]
#define dlam_ra_p       e->con.state[9]
#define dlam_rb_p       e->con.state[10]
#define dlam_rc_p       e->con.state[11]
#define theta_s         e->con.state[12]
#define w_s             e->con.state[13]
#define w_dot_s         e->con.state[14]

#define lam_sa_old          e->con.old_state[0]
#define lam_sb_old          e->con.old_state[1]
#define lam_sc_old          e->con.old_state[2]
#define dlam_sa_old         e->con.old_state[3]
#define dlam_sb_old         e->con.old_state[4]
#define dlam_sc_old         e->con.old_state[5]
#define lam_ra_p_old        e->con.old_state[6]
#define lam_rb_p_old        e->con.old_state[7]
#define lam_rc_p_old        e->con.old_state[8]
#define dlam_ra_p_old       e->con.old_state[9]
#define dlam_rb_p_old       e->con.old_state[10]
#define dlam_rc_p_old       e->con.old_state[11]
#define theta_s_old         e->con.old_state[12]
#define w_s_old             e->con.old_state[13]
#define w_dot_s_old         e->con.old_state[14]

#define RPM                 e->con.ext_out[0]
#define Te                  e->con.ext_out[1]
#define Td                  e->con.ext_out[2]
#define Tl                  e->con.ext_out[3]
#define WATTS               e->con.ext_out[4]
#define HP                  e->con.ext_out[5]
#define Ia                  e->con.ext_out[6]
#define Ib                  e->con.ext_out[7]
#define Ic                  e->con.ext_out[8]


#define TWOPI 6.28319


ind_motor(e,dt)
ELEMENT *e;
double dt;
{

  int i,j;
  double Lls, Llr_p, M, Lms;
  double L11, L21 , L41, L51, L61, L44;
  double cost,costp,costm;

  Ia = i0a;
  Ib = i0b;
```

```
  Ic = i0c;

  RPM = wm * 60.0 / TWOPI;

  cost  = cos(theta);
  costp = cos(theta + TWOPI / 3.0);
  costm = cos(theta - TWOPI / 3.0);

  /* set the default base frequency to 60 HZ if zero */

  if (wbs == 0) wbs = TWOPI * 60.0;

  /* calculate inductances */

  Lls = Xls / wbs;
  Llr_p = Xlr_prime / wbs;
  M = XM / wbs;
  Lms = 2.0 * M / 3.0;

  L11 = Lls + Lms;
  L21 = - Lms / 2.0;

  L44 = Llr_p + Lms;

  L41 = Lms * cost;
  L51 = Lms * costp;
  L61 = Lms * costm;

  /* calculate stator fluxes */

  lam_sa = L11 * i0a + L21 * i0b + L21 * i0c +
    L41 * ira + L51 * irb + L61 * irc;

  lam_sb = L21 * i0a + L11 * i0b + L21 * i0c +
    L61 * ira + L41 * irb + L51 * irc;

  lam_sc = L21 * i0a + L21 * i0b + L11 * i0c +
    L51 * ira + L61 * irb + L41 * irc;

  lam_ra_p = L41 * i0a + L61 * i0b + L51 * i0c +
    L44 * ira + L21 * irb + L21 * irc;

  lam_rb_p = L51 * i0a + L41 * i0b + L61 * i0c +
    L21 * ira + L44 * irb + L21 * irc;

  lam_rc_p = L61 * i0a + L51 * i0b + L41 * i0c +
    L21 * ira + L21 * irb + L44 * irc;

  /* calculate the derivatives of the stator fluxes */

  dlam_sa = v0a - v0n - Rs * i0a;
  dlam_sb = v0b - v0n - Rs * i0b;
  dlam_sc = v0c - v0n - Rs * i0c;
```

```
  dlam_ra_p = -Rr_prime * ira;
  dlam_rb_p = -Rr_prime * irb;
  dlam_rc_p = -Rr_prime * irc;

  /* do the integrations of the fluxes */

  Ila = lam_sa - lam_sa_old - (dt / 2.0) * (dlam_sa + dlam_sa_old);
  Ilb = lam_sb - lam_sb_old - (dt / 2.0) * (dlam_sb + dlam_sb_old);
  Ilc = lam_sc - lam_sc_old - (dt / 2.0) * (dlam_sc + dlam_sc_old);

  Ilra = lam_ra_p - lam_ra_p_old - (dt / 2.0) * (dlam_ra_p + dlam_ra_p_old);
  Ilrb = lam_rb_p - lam_rb_p_old - (dt / 2.0) * (dlam_rb_p + dlam_rb_p_old);
  Ilrc = lam_rc_p - lam_rc_p_old - (dt / 2.0) * (dlam_rc_p + dlam_rc_p_old);

  /* do the current sum */

  Isum = i0a + i0b + i0c;


  /* calculate the torque */

  Te = - pp * Lms * ((i0a * (ira - irb / 2.0 - irc / 2.0) +
                      i0b * (irb - irc / 2.0 - ira / 2.0) +
                      i0c * (irc - ira / 2.0 - irb / 2.0)) * sin(theta) +
                     (sqrt(3.0)/2.0) * cos(theta) *
                     (i0a * (irb - irc) +
                      i0b * (irc - ira) +
                      i0c * (ira - irb)));


  WATTS = Te * wm;
  HP = WATTS / 746.0;

  Td = B * wm;
  Tl = Tmech + Td;

  Torque = Te - J * wm_dt - Tl;

  theta_s = theta;
  w_s = wm;
  w_dot_s = wm_dt;

  /* do the integration of the frequency and acceleration */

  W = theta_s - theta_s_old - (dt / 2.0) * pp * (w_s + w_s_old);
  Wdot = w_s - w_s_old - (dt / 2.0) * (w_dot_s + w_dot_s_old);




}
```

C.7 f_switch_3p.c

f_switch_3p.c

```
/* f_switch_3p.c */
/* Norbert H. Doerry

   12 March 1989

   This routine simulates a 3 phase switch.  When a switch is commanded
   to open, the switch is left closed until the current has a zero-crossing.
   This means that each phase will open at a slightly different time.
   Closing the switch happens instantaneously

*/
#include <stdio.h>
#include <math.h>
#include "doerry.h"

#define v0a e->con.in[0]
#define v0b e->con.in[1]
#define v0c e->con.in[2]
#define v1a e->con.in[3]
#define v1b e->con.in[4]
#define v1c e->con.in[5]
#define i0a e->con.in[6]
#define i0b e->con.in[7]
#define i0c e->con.in[8]
#define i1a e->con.in[9]
#define i1b e->con.in[10]
#define i1c e->con.in[11]
#define sa  e->con.state[0]
#define sb  e->con.state[1]
#define sc  e->con.state[2]
#define ia  e->con.state[3]
#define ib  e->con.state[4]
#define ic  e->con.state[5]
#define sa_old  e->con.old_state[0]
#define sb_old  e->con.old_state[1]
#define sc_old  e->con.old_state[2]
#define ia_old  e->con.old_state[3]
#define ib_old  e->con.old_state[4]
#define ic_old  e->con.old_state[5]
#define v0a_ 0
#define v0b_ 1
#define v0c_ 2
#define v1a_ 3
#define v1b_ 4
#define v1c_ 5
#define i0a_ 6
#define i0b_ 7
#define i0c_ 8
#define i1a_ 9
#define i1b_ 10
#define i1c_ 11
#define Switch e->con.ext_in[0]

switch_3p(e,dt)
```

```
ELEMENT *e;
double dt;
{
  int i,j;
  double va,vb,vc;

  /* initialize the jacobian matrix to zeroes if dt  == 0 */

  for (i = 0 ; dt == 0.0 && i < 6  ; i++)
    for (j = 0 ; j < 12 ; j++)
      e->con.jacob_in[i + 6 * j] = 0.0;

  va = v1a - v0a;
  vb = v1b - v0b;
  vc = v1c - v0c;

  ia = (i1a - i0a) / 2.0;
  ib = (i1b - i0b) / 2.0;
  ic = (i1c - i0c) / 2.0;

  /* see if switch is closed */

  if (Switch == 1.0)
    {
      sa = 1.0;
      sb = 1.0;
      sc = 1.0;
    }
  else  /* if switch is ordered open, use results from last time */
    {
      sa = sa_old;
      sb = sb_old;
      sc = sc_old;
    }

  /* if the switch is closed, the voltage should go to zero, otherwise
     the current should go to zero */

  e->con.implicit[0] = (sa == 1.0) ? va : ia;
  e->con.implicit[1] = (sb == 1.0) ? vb : ib;
  e->con.implicit[2] = (sc == 1.0) ? vc : ic;

  if (sa == 1.0)
    {
      e->con.jacob_in[0 + 6 * v0a_] = - 1.0;
      e->con.jacob_in[0 + 6 * v1a_] =   1.0;
      e->con.jacob_in[0 + 6 * i0a_] =   0.0;
      e->con.jacob_in[0 + 6 * i1a_] =   0.0;
    }
  else
    {
      e->con.jacob_in[0 + 6 * v0a_] =   0.0;
      e->con.jacob_in[0 + 6 * v1a_] =   0.0;
      e->con.jacob_in[0 + 6 * i0a_] = - 0.5;
```

```
      e->con.jacob_in[0 + 6 * i1a_] =   0.5;
    }

  if (sb == 1.0)
    {
      e->con.jacob_in[1 + 6 * v0b_] = - 1.0;
      e->con.jacob_in[1 + 6 * v1b_] =   1.0;
      e->con.jacob_in[1 + 6 * i0b_] =   0.0;
      e->con.jacob_in[1 + 6 * i1b_] =   0.0;
    }
  else
    {
      e->con.jacob_in[1 + 6 * v0b_] =   0.0;
      e->con.jacob_in[1 + 6 * v1b_] =   0.0;
      e->con.jacob_in[1 + 6 * i0b_] = - 0.5;
      e->con.jacob_in[1 + 6 * i1b_] =   0.5;
    }

  if (sc == 1.0)
    {
      e->con.jacob_in[2 + 6 * v0c_] = - 1.0;
      e->con.jacob_in[2 + 6 * v1c_] =   1.0;
      e->con.jacob_in[2 + 6 * i0c_] =   0.0;
      e->con.jacob_in[2 + 6 * i1c_] =   0.0;
    }
  else
    {
      e->con.jacob_in[2 + 6 * v0c_] =   0.0;
      e->con.jacob_in[2 + 6 * v1c_] =   0.0;
      e->con.jacob_in[2 + 6 * i0c_] = - 0.5;
      e->con.jacob_in[2 + 6 * i1c_] =   0.5;
    }

  /* the last three implicit variables assure currents are the same */

  e->con.implicit[3] = i0a + i1a;
  e->con.implicit[4] = i0b + i1b;
  e->con.implicit[5] = i0c + i1c;

  e->con.jacob_in[3 + 6 * i0a_] = 1.0;
  e->con.jacob_in[3 + 6 * i1a_] = 1.0;
  e->con.jacob_in[4 + 6 * i0b_] = 1.0;
  e->con.jacob_in[4 + 6 * i1b_] = 1.0;
  e->con.jacob_in[5 + 6 * i0c_] = 1.0;
  e->con.jacob_in[5 + 6 * i1c_] = 1.0;

  /* see if should open the switches (look for zero crossing) */

  if (Switch == 0)
    {
      sa = (ia * ia_old <= 0.0) ? 0.0 : 1.0;
      sb = (ib * ib_old <= 0.0) ? 0.0 : 1.0;
      sc = (ic * ic_old <= 0.0) ? 0.0 : 1.0;
    }
```

```
  /* turn the jacob_switch on */

  e->con.jacob_switch = 1;

  /* save the external output variables */

  for (i = 0 ; i < 6 ; i++)
    e->con.ext_out[i] = e->con.state[i];


}
```

C.8 f_breaker_3p.c

f_breaker_3p.c

```
/* f_breaker_3p.c */
/* Norbert H. Doerry

   8 April 1989

   This routine simulates a 3 phase breaker.  When a breaker is commanded
   to open, the switch is left closed until the current has a zero-crossing.
   This means that each phase will open at a slightly different time.
   Closing the switch happens instantaneously.

   The overcurrent switch is based on a psuedo-rms value for the current


*/
#include <stdio.h>
#include <math.h>
#include "doerry.h"

#define v0a e->con.in[0]
#define v0b e->con.in[1]
#define v0c e->con.in[2]
#define v1a e->con.in[3]
#define v1b e->con.in[4]
#define v1c e->con.in[5]
#define i0a e->con.in[6]
#define i0b e->con.in[7]
#define i0c e->con.in[8]
#define i1a e->con.in[9]
#define i1b e->con.in[10]
#define i1c e->con.in[11]
#define sa  e->con.state[0]
#define sb  e->con.state[1]
#define sc  e->con.state[2]
#define ia  e->con.state[3]
#define ib  e->con.state[4]
#define ic  e->con.state[5]
#define ave_ia e->con.state[6]
#define ave_ib e->con.state[7]
#define ave_ic e->con.state[8]
#define t_ia   e->con.state[9]
#define t_ib   e->con.state[10]
#define t_ic   e->con.state[11]
#define sa_old  e->con.old_state[0]
#define sb_old  e->con.old_state[1]
#define sc_old  e->con.old_state[2]
#define ia_old  e->con.old_state[3]
#define ib_old  e->con.old_state[4]
#define ic_old  e->con.old_state[5]
#define ave_ia_old e->con.old_state[6]
#define ave_ib_old e->con.old_state[7]
#define ave_ic_old e->con.old_state[8]
#define t_ia_old   e->con.old_state[9]
#define t_ib_old   e->con.old_state[10]
#define t_ic_old   e->con.old_state[11]
```

f_breaker_3p.c

```
#define v0a_ 0
#define v0b_ 1
#define v0c_ 2
#define v1a_ 3
#define v1b_ 4
#define v1c_ 5
#define i0a_ 6
#define i0b_ 7
#define i0c_ 8
#define i1a_ 9
#define i1b_ 10
#define i1c_ 11
#define Switch e->con.ext_in[0]

#define f           e->con.param[0]
#define I_trip      e->con.param[1]
#define time_trip   e->con.param[2]


/* The following are the states of the switches */

#define ALL_ON      7.0
#define WRONG       6.0
#define TRIPPED_C   5.0
#define TRIPPED_O   4.0
#define SW_OFF_C    3.0
#define SW_OFF_O    2.0
#define ALL_OFF_C   1.0
#define ALL_OFF_O   0.0


breaker_3p(e,dt)
ELEMENT *e;
double dt;
{
  int i,j;
  double va,vb,vc;

  /* initialize the jacobian matrix to zeroes if dt  == 0 */

  for (i = 0 ; dt == 0.0 && i < 6  ; i++)
    for (j = 0 ; j < 12 ; j++)
      e->con.jacob_in[i + 6 * j] = 0.0;

  va = v1a - v0a;
  vb = v1b - v0b;
  vc = v1c - v0c;

  ia = (i1a - i0a) / 2.0;
  ib = (i1b - i0b) / 2.0;
  ic = (i1c - i0c) / 2.0;

  /* see if modes are illegal */
```

```
  {
    if (sa > 7 || sa < 0 || sa == WRONG) sa = ALL_ON;
    if (sb > 7 || sb < 0 || sb == WRONG) sb = ALL_ON;
    if (sc > 7 || sc < 0 || sc == WRONG) sc = ALL_ON;
  }

  if (Switch == 1.0)
    {
      f_switch_on(&sa);
      f_switch_on(&sb);
      f_switch_on(&sc);
    }
  else
    {
      f_switch_off(&sa);
      f_switch_off(&sb);
      f_switch_off(&sc);
    }

  /* see if breaker should open */

  if (t_ia_old >= time_trip || t_ib_old >= time_trip || t_ic_old >= time_trip)
    {
      f_breaker_off(&sa);
      f_breaker_off(&sb);
      f_breaker_off(&sc);
    }


  /* if the switch is closed, the voltage should go to zero, otherwise
     the current should go to zero */

  e->con.implicit[0] = ((int) sa % 2 == 1.0) ? va : ia;
  e->con.implicit[1] = ((int) sb % 2 == 1.0) ? vb : ib;
  e->con.implicit[2] = ((int) sc % 2 == 1.0) ? vc : ic;

  if ((int) sa % 2 == 1.0)
    {
      e->con.jacob_in[0 + 6 * v0a_] = - 1.0;
      e->con.jacob_in[0 + 6 * v1a_] =   1.0;
      e->con.jacob_in[0 + 6 * i0a_] =   0.0;
      e->con.jacob_in[0 + 6 * i1a_] =   0.0;
    }
  else
    {
      e->con.jacob_in[0 + 6 * v0a_] =   0.0;
      e->con.jacob_in[0 + 6 * v1a_] =   0.0;
      e->con.jacob_in[0 + 6 * i0a_] = - 0.5;
      e->con.jacob_in[0 + 6 * i1a_] =   0.5;
    }

  if ((int) sb % 2 == 1.0)
    {
      e->con.jacob_in[1 + 6 * v0b_] = - 1.0;
```

```
      e->con.jacob_in[1 + 6 * v1b_] =   1.0;
      e->con.jacob_in[1 + 6 * i0b_] =   0.0;
      e->con.jacob_in[1 + 6 * i1b_] =   0.0;
    }
  else
    {
      e->con.jacob_in[1 + 6 * v0b_] =   0.0;
      e->con.jacob_in[1 + 6 * v1b_] =   0.0;
      e->con.jacob_in[1 + 6 * i0b_] = - 0.5;
      e->con.jacob_in[1 + 6 * i1b_] =   0.5;
    }

  if ((int) sc % 2 == 1.0)
    {
      e->con.jacob_in[2 + 6 * v0c_] = - 1.0;
      e->con.jacob_in[2 + 6 * v1c_] =   1.0;
      e->con.jacob_in[2 + 6 * i0c_] =   0.0;
      e->con.jacob_in[2 + 6 * i1c_] =   0.0;
    }
  else
    {
      e->con.jacob_in[2 + 6 * v0c_] =   0.0;
      e->con.jacob_in[2 + 6 * v1c_] =   0.0;
      e->con.jacob_in[2 + 6 * i0c_] = - 0.5;
      e->con.jacob_in[2 + 6 * i1c_] =   0.5;
    }

  /* the last three implicit variables assure currents are the same */

  e->con.implicit[3] = i0a + i1a;
  e->con.implicit[4] = i0b + i1b;
  e->con.implicit[5] = i0c + i1c;

  e->con.jacob_in[3 + 6 * i0a_] = 1.0;
  e->con.jacob_in[3 + 6 * i1a_] = 1.0;
  e->con.jacob_in[4 + 6 * i0b_] = 1.0;
  e->con.jacob_in[4 + 6 * i1b_] = 1.0;
  e->con.jacob_in[5 + 6 * i0c_] = 1.0;
  e->con.jacob_in[5 + 6 * i1c_] = 1.0;

  /* see if should open the switches (look for zero crossing) */

  if (ia * ia_old <= 0.0) f_zero_cross(&sa);
  if (ib * ib_old <= 0.0) f_zero_cross(&sb);
  if (ic * ic_old <= 0.0) f_zero_cross(&sc);

  /* update the current timers */

  /* calculate the rms currents */

  if (f > 1.0) f = 1.0;
  if (f < 0) f = 0.0;

  /* compute new average */
```

```
  ave_ia = sqrt( (1.0 - f) * ia * ia + f * ave_ia_old * ave_ia_old);
  ave_ib = sqrt( (1.0 - f) * ib * ib + f * ave_ib_old * ave_ib_old);
  ave_ic = sqrt( (1.0 - f) * ic * ic + f * ave_ic_old * ave_ic_old);

  t_ia = (ave_ia >= I_trip) ? t_ia_old += dt : 0;
  t_ib = (ave_ib >= I_trip) ? t_ib_old += dt : 0;
  t_ic = (ave_ic >= I_trip) ? t_ic_old += dt : 0;



  /* turn the jacob_switch on */

  e->con.jacob_switch = 1;

  /* save the external output variables */

  for (i = 0 ; i < 12 ; i++)
    e->con.ext_out[i] = e->con.state[i];



}

/* f_switch_on performs the transformation of states for the
   external input variable turning on */

f_switch_on(x)
double *x;
{
  if       (*x == ALL_OFF_O) *x = ALL_ON;
  else if  (*x == ALL_OFF_C) *x = ALL_ON;
  else if  (*x == SW_OFF_O)  *x = ALL_ON;
  else if  (*x == SW_OFF_C)  *x = ALL_ON;
  else if  (*x == TRIPPED_O) return;
  else if  (*x == TRIPPED_C) return;
  else if  (*x == ALL_ON) return;
  else      *x = ALL_ON;
}

f_switch_off(x)
double *x;
{
  if       (*x == ALL_OFF_O) *x = ALL_OFF_O;
  else if  (*x == ALL_OFF_C) *x = ALL_OFF_C;
  else if  (*x == SW_OFF_O)  *x = SW_OFF_O;
  else if  (*x == SW_OFF_C)  *x = SW_OFF_C;
  else if  (*x == TRIPPED_O) *x = ALL_OFF_O;
  else if  (*x == TRIPPED_C) *x = ALL_OFF_C;
  else if  (*x == ALL_ON)    *x = SW_OFF_C;
  else      *x = SW_OFF_C;
}

f_breaker_off(x)
```

f_breaker_3p.c

```
double *x;
{
  if       (*x == ALL_OFF_O) *x = ALL_OFF_O;
  else if  (*x == ALL_OFF_C) *x = ALL_OFF_C;
  else if  (*x == SW_OFF_O)  *x = ALL_OFF_O;
  else if  (*x == SW_OFF_C)  *x = ALL_OFF_C;
  else if  (*x == TRIPPED_O) *x = TRIPPED_O;
  else if  (*x == TRIPPED_C) *x = TRIPPED_C;
  else if  (*x == ALL_ON)    *x = TRIPPED_C;
  else      *x = TRIPPED_C;
}


f_zero_cross(x)
double *x;
{
  if       (*x == ALL_OFF_O) *x = ALL_OFF_O;
  else if  (*x == ALL_OFF_C) *x = ALL_OFF_O;
  else if  (*x == SW_OFF_O)  *x = SW_OFF_O;
  else if  (*x == SW_OFF_C)  *x = SW_OFF_O;
  else if  (*x == TRIPPED_O) *x = TRIPPED_O;
  else if  (*x == TRIPPED_C) *x = TRIPPED_O;
  else if  (*x == ALL_ON)    *x = ALL_ON;
  else      *x = ALL_ON;
}
```

C.9 f_spst_switch.c

```
/* f_spst_switch.c */
/* Norbert H. Doerry

   15 March 1989

   This file describes a switch

*/

#include <stdio.h>
#include <math.h>
#include "doerry.h"


#define v0 e->con.in[0]
#define v1 e->con.in[1]
#define i0 e->con.in[2]
#define i1 e->con.in[3]
#define v0_ 0
#define v1_ 1
#define i0_ 2
#define i1_ 3
#define Switch e->con.ext_in[0]
#define v      e->con.ext_out[0]
#define i      e->con.ext_out[1]

spst_switch(e,dt)
ELEMENT *e;
double dt;
{
  double integ();


  v = v0 - v1;
  i = (i0 - i1) / 2.0;


  /* second implicit variable is sum of currents */

  e->con.implicit[1] = i0 + i1;

  /* set up the jacobian matrix */

  e->con.jacob_switch = 1;   /* turn jacobian switch on */

  if (Switch == 0) /* open */
    {
      e->con.implicit[0] = i;

      e->con.jacob_in[0 + 2 * v0_] =  0.0;
      e->con.jacob_in[0 + 2 * v1_] =  0.0;
      e->con.jacob_in[0 + 2 * i0_] =  0.5;
      e->con.jacob_in[0 + 2 * i1_] = -0.5;
    }
```

```
  else  /* closed */
    {
      e->con.implicit[0] = v;

      e->con.jacob_in[0 + 2 * v0_] =    1.0;
      e->con.jacob_in[0 + 2 * v1_] =  - 1.0;
      e->con.jacob_in[0 + 2 * i0_] =    0.0;
      e->con.jacob_in[0 + 2 * i1_] =    0.0;
    }


  e->con.jacob_in[1 + 2 * v0_] = 0.0;
  e->con.jacob_in[1 + 2 * v1_] = 0.0;
  e->con.jacob_in[1 + 2 * i0_] = 1.0;
  e->con.jacob_in[1 + 2 * i1_] = 1.0;

}
```

C.10 f_gen_synch_3p.c

f_gen_synch_3p.c

```
/* f_gen_synch_3p.c */
/* Norbert H. Doerry */
/* 12 March 1989

   This file simulates a synchronous generator that is modelled as
   a three phase source with a series inductance.  The magnitude and
   the frequency of the generator are input variables.  The phase angle
   of phase a along with the inductance are parameters.

*/
#include <stdio.h>
#include <math.h>
#include "doerry.h"


#define RAD_DEG     0.017453293
#define DEG_RAD    57.29578
#define HZ_RADSEC   0.15915494
#define RADSEC_HZ   6.2831853
#define PHASE_SHIFT 2.0943951

#define v0a e->con.in[0]
#define v0b e->con.in[1]
#define v0c e->con.in[2]
#define v0n e->con.in[3]
#define i0a e->con.in[4]
#define i0b e->con.in[5]
#define i0c e->con.in[6]
#define i0n e->con.in[7]
#define Vmag e->con.in[8]
#define freq e->con.in[9]
#define va  e->con.state[0]
#define vb  e->con.state[1]
#define vc  e->con.state[2]
#define ia  e->con.state[3]
#define ib  e->con.state[4]
#define ic  e->con.state[5]
#define t   e->con.state[6]
#define vga e->con.state[7]
#define vgb e->con.state[8]
#define vgc e->con.state[9]
#define va_old  e->con.old_state[0]
#define vb_old  e->con.old_state[1]
#define vc_old  e->con.old_state[2]
#define ia_old  e->con.old_state[3]
#define ib_old  e->con.old_state[4]
#define ic_old  e->con.old_state[5]
#define t_old   e->con.old_state[6]
#define vga_old e->con.old_state[7]
#define vgb_old e->con.old_state[8]
#define vgc_old e->con.old_state[9]
#define v0a_ 0
#define v0b_ 1
#define v0c_ 2
```

```
#define v0n_ 3
#define i0a_ 4
#define i0b_ 5
#define i0c_ 6
#define i0n_ 7
#define Vmag_ 8
#define freq_ 9
#define phase_a  e->con.param[0]
#define L  e->con.param[1]
#define R  e->con.param[2]

gen_synch_3p(e,dt)
ELEMENT *e;
double dt;
{
  int i,j;
  double fta,ftb,ftc,fta_old,ftb_old,ftc_old;
  double pa,pb,pc;

  /* initialize the jacobian matrix to zeroes if dt  == 0 */

  for (i = 0 ; dt == 0.0  && i < 4 ; i++)
    for (j = 0 ; j < 10 ; j++)
      e->con.jacob_in[i + 4 * j] = 0.0;

  /* calculate states */

  va = v0a - v0n;
  vb = v0b - v0n;
  vc = v0c - v0n;
  ia = (i0a - i0n - i0b - i0c) / 2.0;
  ib = (i0b - i0n - i0c - i0a) / 2.0;
  ic = (i0c - i0n - i0a - i0b) / 2.0;

  /* update the time counter */

  t = dt + t_old;

  /* calculate phases */

  pa = (freq * t * RADSEC_HZ + phase_a * RAD_DEG);
  pb = pa - PHASE_SHIFT;
  pc = pa + PHASE_SHIFT;

  /* calculate phase generator voltages */

  vga = Vmag * cos(pa);
  vgb = Vmag * cos(pb);
  vgc = Vmag * cos(pc);


  /* sum of currents should be zero */

  e->con.implicit[3] = i0a + i0b + i0c + i0n;
```

```
  e->con.jacob_in[3 + 4 * i0a_] = 1.0;
  e->con.jacob_in[3 + 4 * i0b_] = 1.0;
  e->con.jacob_in[3 + 4 * i0c_] = 1.0;
  e->con.jacob_in[3 + 4 * i0n_] = 1.0;

  /* see if inductance is zero */

  if (L == 0)
    {
      e->con.implicit[0] = va - vga;
      e->con.implicit[1] = vb - vgb;
      e->con.implicit[2] = vc - vgc;

      e->con.jacob_in[0 + 4 * v0a_] =  1.0;
      e->con.jacob_in[0 + 4 * v0n_] = -1.0;
      e->con.jacob_in[0 + 4 * freq_] = Vmag * t * RADSEC_HZ * sin(pa);
      e->con.jacob_in[0 + 4 * Vmag_] = - cos(pa);

      e->con.jacob_in[1 + 4 * v0b_] =  1.0;
      e->con.jacob_in[1 + 4 * v0n_] = -1.0;
      e->con.jacob_in[1 + 4 * freq_] = Vmag * t * RADSEC_HZ * sin(pb);
      e->con.jacob_in[1 + 4 * Vmag_] = - cos(pb);

      e->con.jacob_in[2 + 4 * v0c_] =  1.0;
      e->con.jacob_in[2 + 4 * v0n_] = -1.0;
      e->con.jacob_in[2 + 4 * freq_] = Vmag * t * RADSEC_HZ * sin(pc);
      e->con.jacob_in[2 + 4 * Vmag_] = - cos(pc);
    }
  else if (R == 0) /* if leakage resistance is zero, output voltages are 0 */
    {
      e->con.implicit[0] = va;
      e->con.implicit[1] = vb;
      e->con.implicit[2] = vc;

      e->con.jacob_in[0 + 4 * v0a_] =  1.0;
      e->con.jacob_in[0 + 4 * v0n_] = -1.0;

      e->con.jacob_in[1 + 4 * v0b_] =  1.0;
      e->con.jacob_in[1 + 4 * v0n_] = -1.0;

      e->con.jacob_in[2 + 4 * v0c_] =  1.0;
      e->con.jacob_in[2 + 4 * v0n_] = -1.0;
    }
  else

    {

      e->con.implicit[0] = ia - ia_old - (dt/2.0) *
        ((va - vga) + (va_old - vga_old)) / L - (va - va_old) / R;
      e->con.implicit[1] = ib - ib_old - (dt/2.0) *
        ((vb - vgb) + (vb_old - vgb_old)) / L - (va - va_old) / R;
      e->con.implicit[2] = ic - ic_old - (dt/2.0) *
        ((vc - vgc) + (vc_old - vgc_old)) / L - (va - va_old) / R;
```

```
      e->con.jacob_in[0 + 4 * i0a_] =  0.5;
      e->con.jacob_in[0 + 4 * i0b_] = -0.5;
      e->con.jacob_in[0 + 4 * i0c_] = -0.5;
      e->con.jacob_in[0 + 4 * i0n_] = -0.5;
      e->con.jacob_in[0 + 4 * v0a_] = - dt / (2.0 * L) - 1.0 / R;
      e->con.jacob_in[0 + 4 * v0n_] = - e->con.jacob_in[0 + 4 * v0a_];
      e->con.jacob_in[0 + 4 * Vmag_]= cos(pa) * dt / (2.0 * L) ;
      e->con.jacob_in[0 + 4 * freq_]=
        - dt * Vmag * sin(pa) * t * RADSEC_HZ / (2.0 * L);

      e->con.jacob_in[1 + 4 * i0b_] =  0.5;
      e->con.jacob_in[1 + 4 * i0c_] = -0.5;
      e->con.jacob_in[1 + 4 * i0a_] = -0.5;
      e->con.jacob_in[1 + 4 * i0n_] = -0.5;
      e->con.jacob_in[1 + 4 * v0b_] = - dt / (2.0 * L) - 1.0 / R;
      e->con.jacob_in[1 + 4 * v0n_] = - e->con.jacob_in[1 + 4 * v0b_];
      e->con.jacob_in[1 + 4 * Vmag_]= cos(pb) * dt / (2.0 * L) ;
      e->con.jacob_in[1 + 4 * freq_]=
        - dt * Vmag * sin(pb) * t * RADSEC_HZ / (2.0 * L);

      e->con.jacob_in[2 + 4 * i0c_] =  0.5;
      e->con.jacob_in[2 + 4 * i0a_] = -0.5;
      e->con.jacob_in[2 + 4 * i0b_] = -0.5;
      e->con.jacob_in[2 + 4 * i0n_] = -0.5;
      e->con.jacob_in[2 + 4 * v0c_] = - dt / (2.0 * L) - 1.0 / R;
      e->con.jacob_in[2 + 4 * v0n_] = - e->con.jacob_in[2 + 4 * v0c_];
      e->con.jacob_in[2 + 4 * Vmag_]= cos(pc) * dt / (2.0 * L) ;
      e->con.jacob_in[2 + 4 * freq_]=
        - dt * Vmag * sin(pc) * t * RADSEC_HZ / (2.0 * L);

    }

  /* turn the jacob switch on */

  e->con.jacob_switch = 1;

  /* store external output variables */

  for ( i = 0 ; i < 6 ; i++)
    e->con.ext_out[i] = e->con.state[i];


}
```

C.11 penner.h

```
/* penner.h */
/* Norbert H. Doerry

   14  March 1989

   This is an include file which tells the main program where to get
   the proper information for the devices

   *** Modified 11 April 1989 by nhd ****

   added breaker_3p

   *** Modified 15 April 1989 by nhd ****

   added  synch_mach, speed_reg, volt_reg,ind_motor,gas_turbine, source
          integrator

  *** Modified 27 April 1989  by nhd ****

  added volt_meter

*/


typedef int (*FUNCTION_PTR)();

#define NBR_DEV_FILES 2          /* number of device description files */

static char *device_file[] =     /* names of the device description files */
{
  "/mit/13.411/sepsip/three_phase.input",
  "/mit/13.411/sepsip/one_phase.input"
};

static int nbr_device_file[] =
{
  12,                              /* number of devices per file */
  10
};

static char *device_name[] =        /* names of devices */
{
  "t_line_3p",
  "rl_wye",
  "gen_synch_3p",
  "switch_3p",
  "rms",
  "breaker_3p",
  "synch_mach",
  "speed_reg",
  "volt_reg",
  "ind_motor",
  "gas_turbine",
```

```
  "volt_meter",

  "inductor",
  "capacitor",
  "resistor",
  "voltage_source",
  "current_source",
  "diode",
  "switch",
  "pulse_switch",
  "source",
  "integrator"
};

/* device functions for the above device names */

#define F0 t_line_3p
#define F1 rl_wye
#define F2 gen_synch_3p
#define F3 switch_3p
#define F4 rms
#define F4a breaker_3p
#define F4b synch_mach
#define F4c speed_reg
#define F4d volt_reg
#define F4e ind_motor
#define F4f gas_turbine
#define F4g volt_meter

#define F5 inductor
#define F6 capacitor
#define F7 resistor
#define F8 voltage_source
#define F9 current_source
#define F10 diode
#define F11 spst_switch
#define F12 pulse_switch
#define F13 source
#define F14 integrator

int F0 ();
int F1 ();
int F2 ();
int F3 ();
int F4 ();
int F4a();
int F4b();
int F4c();
int F4d();
int F4e();
int F4f();
int F4g();
int F5 ();
int F6 ();
```

```
int F7 ();
int F8 ();
int F9 ();
int F10 ();
int F11 ();
int F12 ();
int F13 ();
int F14 ();


static FUNCTION_PTR dev_fnctn[]  =     /* addresses of device functions */
{
  F0,
  F1,
  F2,
  F3,
  F4,
  F4a,
  F4b,
  F4c,
  F4d,
  F4e,
  F4f,
  F4g,
  F5,
  F6,
  F7,
  F8,
  F9,
  F10,
  F11,
  F12,
  F13,
  F14

};
```

C.12 three_phase.input

```
! three_phase.input
!
! Norbert H Doerry
!  11 March 1989
!  *** last update  17 April 1989
!  *** updated      27 April 1989
!
! This file describes the following devices
!
!     t_line_3p     :transmission line
!     rl_wye        :RL Wye Load
!     gen_synch_3p  :synchronous generator with synchronous reactance
!     switch_3p     :three phase switch
!     rms           :calculates average value of voltage
!     breaker       :three phase breaker
!     synch_mach    :synchronous machine model
!     speed_reg     :speed regulator model (turbine with governor)
!     volt_reg      :voltage regulator (field excitation)
!     ind_motor     :induction motor model
!     gas_turbine   :gas_turbine model
!     volt_meter    :three phase voltage meter
!
name t_line_3p
inputs 12
   v0a
   v0b
   v0c
   v1a
   v1b
   v1c
   i0a
   i0b
   i0c
   i1a
   i1b
   i1c
states 6
   va
   vb
   vc
   ia
   ib
   ic
implicit 6
   integ_a
   integ_b
   integ_c
   i_sum_a
   i_sum_b
   i_sum_c
external output 6
   float Va
   float Vb
   float Vc
```

```
    float Ia
    float Ib
    float Ic
 parameters 3
    R
    L
    R1
 end
 !
 name rl_wye
 inputs 8
    v0a
    v0b
    v0c
    v0n
    i0a
    i0b
    i0c
    i0n
 states 6
    va
    vb
    vc
    ia
    ib
    ic
implicit 4
    int_a
    int_b
    int_c
    i_sum
external output 6
    float Va
    float Vb
    float Vc
    float Ia
    float Ib
    float Ic
parameters 3
    R
    L
    R1
end
!
name gen_synch_3p
inputs 10
    v0a
    v0b
    v0c
    v0n
    i0a
    i0b
    i0c
    i0n
```

```
   Vmag
   freq
states 10
   va
   vb
   vc
   ia
   ib
   ic
   t
   vga
   vgb
   vgc
implicit 4
   int_a
   int_b
   int_c
   i_sum
external output 6
   float Va
   float Vb
   float Vc
   float Ia
   float Ib
   float Ic
parameters 3
   phase_a
   L
   R
end
!
name switch_3p
inputs 12
   v0a
   v0b
   v0c
   v1a
   v1b
   v1c
   i0a
   i0b
   i0c
   i1a
   i1b
   i1c
states 6
   sa
   sb
   sc
   ia
   ib
   ic
implicit 6
   integ_a
```

```
    integ_b
    integ_c
    i_sum_a
    i_sum_b
    i_sum_c
external input 1
    switch Switch
external output 6
    switch Sa
    switch Sb
    switch Sc
    float Ia
    float Ib
    float Ic
end
!
name rms
inputs 2
    v0
    v1
states 1
    ave
external output 1
    float v
parameters 1
    f
end
!
name breaker_3p
inputs 12
    v0a
    v0b
    v0c
    v1a
    v1b
    v1c
    i0a
    i0b
    i0c
    i1a
    i1b
    i1c
states 12
    sa
    sb
    sc
    ia
    ib
    ic
    ave_ia
    ave_ib
    ave_ic
    t_ia
    t_ib
```

```
three_phase.input


      t_ic
   implicit 6
      integ_a
      integ_b
      integ_c
      i_sum_a
      i_sum_b
      i_sum_c
   external input 1
      switch Switch
   external output 6
      switch Sa
      switch Sb
      switch Sc
      float Ia
      float Ib
      float Ic
   parameters 3
      f
      I_trip
      time_trip
   end
   !
   name synch_mach
   inputs 17
      v0a
      v0b
      v0c
      v0n
      i0a
      i0b
      i0c
      v0f
      v1f
      i0f
      i1f
      theta
      wm
      wm_dt
      Te
      Psi_q
      Psi_d
   states 13
      s_theta
      s_wm
      s_wm_dt
      psi_d
      psi_q
      eq_p
      eq_pp
      ed_pp
      d_psi_d
      d_psi_q
      d_eq_p
```

```
    d_eq_pp
    d_ed_pp
 implicit 11
    isum
    ifsum
    vo
    i_psi_d
    i_psi_q
    i_eq_pp
    i_ed_pp
    i_eq_p
    Torq
    W
    Wdot
 parameters 16
    xd
    xq
    xd_p
    xd_pp
    xq_pp
    xal
    Tdo_p
    Tdo_pp
    Tqo_pp
    Tad
    Ifnl
    H
    pp
    wbs
    Vdb
    Pbs
external output 24
    float xad
    float xkd
    float xf
    float rf
    float IdB
    float IfB
    float VfB
    float Tbs
    float alpha
    float ifd
    float vfd
    float eaf
    float vd
    float vq
    float id
    float iq
    float Tepu
    float RPM
    float Pmech
    float Pe
    float Tacc
    float Ia
```

```
   float Ib
   float Ic
end
name speed_reg
inputs 3
  wm
  Tm
  s
states 2
!  wg
!  ws
  Tm_order
  Tmpu
implicit 1
  dT_dt
parameters 6
  wnlo
  wds
  wdTepu
  TBS
  Tg
  B
external outputs 4
  float Tmm
  float Tm_
  float Pshaft
  float Pdeliver
end
name volt_reg
inputs 11
  v0a
  v0b
  v0c
  v0f
  v1f
  i0f
  i1f
  vbs
  wbs
  phase
  vt
states 4
  Verr
  Vsig
  theta
  clip
parameters 5
  Vfdbs
  K
  Tvr
  Vfmax
  Vfmin
implicit 4
  I1
```

```
    I2
    Isum
    Integrate
  end
  name ind_motor
  inputs 13
    v0a
    v0b
    v0c
    i0a
    i0b
    i0c
    theta
    wm
    wm_dt
    v0n
    ira
    irb
    irc
  parameters 9
    Rs
    Xls
    XM
    Xlr_prime
    Rr_prime
    J
    wbs
    pp
    B
  states 15
    lam_sa
    lam_sb
    lam_sc
    dlam_sa
    dlam_sb
    dlam_sc
    lam_ra_p
    lam_rb_p
    lam_rc_p
    dlam_ra_p
    dlam_rb_p
    dlam_rc_p
    theta_s
    w_s
    w_dot_s
  implicit 10
    Ila
    Ilb
    Ilc
    Ilra
    Ilrb
    Ilrc
    Isum
    W
```

```
  Wdot
  Torque
external inputs 1
  float Tmech
external outputs 9
  float RPM
  float Te
  float Td
  float Tl
  float WATTS
  float HP
  float Ia
  float Ib
  float Ic
end
name gas_turbine
inputs 3
  wm
  wm_dt
  Tm
end
name volt_meter
inputs 6
  v0a
  v0b
  v0c
  wbs
  phase
  vt
parameters 1
  Vbs
implicit 2
  I1
  I2
states  1
  theta
external outputs 2
  float Vt
  float Phase
end
```

C.13 one_phase.input

```
! one_phase.input
!
! Norbert H Doerry
!
! 15 March 1989/ modified 15 april 1989
!
!  The descriptions for the following devices are contained here
!     inductor
!     capacitor
!     resistor
!     voltage_source
!     current_source
!     diode
!     switch
!     pulse_switch
!     source
!     integrator
!
name inductor
inputs 4
  v0
  v1
  i0
  i1
states 2
  v
  i
implicit 2
  integrator
  current_sum
external output 2
  float v
  float i
parameters 1
  L
end
!
!
name capacitor
inputs 4
  v0
  v1
  i0
  i1
states 2
  v
  i
implicit 2
  integrator
  current_sum
external output 2
  float v
  float i
parameters 1
```

```
  C
end
!
!
name resistor
inputs 4
  v0
  v1
  i0
  i1
implicit 2
  ohms_law
  current_sum
external output 2
  float v
  float i
parameters 1
  R
end
!
!
name voltage_source
inputs 4
   v0
   v1
   i0
   i1
implicit 2
   voltage_difference
   current_sum
external output 2
   float v
   float i
external input 1
   float V0
end
#
#
name current_source
inputs 4
   v0
   v1
   i0
   i1
implicit 2
   current_0
   current_1
external output 2
   float v
   float i
external input 1
   float I0
end
#
```

one_phase.input

```
#
name diode
inputs 4
  v0
  v1
  i0
  i1
implicit 2
  diode_law
  current_sum
external output 2
  float v
  float i
parameters 1
  Vd
end
!
!
name switch
inputs 4
  v0
  v1
  i0
  i1
implicit 2
  switch_eqt
  current_sum
external output 2
  float v
  float i
external input 1
  switch switch
end
!
!
name pulse_switch
inputs 4
  v0
  v1
  i0
  i1
implicit 2
  switch_eqt
  current_sum
states 1
  time
external output 3
  float v
  float i
  switch switch
parameters 4
  period
  duty_cycle
  t_on
```

one_phase.input

```
  t_off
end
!
name source
inputs 1
  out
implicit 1
  difference
external input 1
  float in
parameters 1
  scale
end
name integrator
inputs 2
  X
  Y
implicit 1
  integ
parameters 1
  tau
states 2
  X
  Y
end
```

APPENDIX D

MENU DRIVER CODE

D.1 menu.c

SEPSIP uses a separate menu driver program for the utility menu. This was done to allow the user to customize the utility options without recompiling SEPSIP. By editing the file **sepsip_util.menu**, the utility menu can be modified without any recompilation.

```
/* menu.c */
/* Norbert H Doerry
   5 March 1988
   This program is a universal menu driver.  It requires
   that another file exist with the same name plus an
   extension of .menu   This second file is a text file
   that contains the program header and all the menu
   information in the following format :

   ( header ) ( As many lines as you would like as long
                as none of them begins with !)
   !  (header delimiter)
   string  (what the user types in to execute the
                corresponding menu item)
   name  (name of menu item)
   command  (program or command to be executed)
   ...  (This 3 line sequence repeated)
   !  (command delimiter)

   If more than one command is given the same character,
   the first one on the list will be executed.

   The program also puts the following command at the
   top of the list :
   q
   Quit
   exit()

   Thus you can't use a q or ! for a character

   A ! inputted from the user will repeat the last
   command executed.

   ***** VERSION 2.0 ********

   Heavily revised to include the following
      - commands can be up to 10 characters long
      - arguments can be passed to the commands
      - arguments can be passed to the program


*/
#include <stdio.h>
#define CHRLEN 81
#define ARRLEN 21

typedef struct Cmd
{
  char string[CHRLEN];
  char name[CHRLEN];
  char command[CHRLEN];
  struct Cmd *next;
}
CMD;
```

```
main(argc,argv)
int argc;
char **argv;
{
  char comm[CHRLEN];       /* name of this program (minus path)*/
  char command[CHRLEN];    /* name of this program (with path) */
  char inline[CHRLEN];
  char line[CHRLEN];
  char outline[CHRLEN];
  char last_command[CHRLEN];
  char last_all[CHRLEN];
  int i,j,numcmd,flag,len;
  FILE *in;
  char ch;
  CMD cmd,*cptr;
  char *calloc();


  strcpy(command,argv[0]);
  strcat(command,".menu");
  len = strlen(argv[0]);
  for (i=len; ((i != -1) && argv[0][i] != '/') ; i--);
  i++;
  for ( j = 0 ; i <= len ; i++)
    comm[j++] = argv[0][i];


  if ((in = fopen(command,"r")) == NULL)
    {
      printf("Can't find %s\n",command);
      exit();
    }
  inline[0] = NULL;
  while ((fgets(inline,CHRLEN,in) != NULL) && inline[0] != '!')
    {
      inline[(strlen(inline) - 1)] = NULL;
      puts(inline);
    }

  cptr = &cmd;
  cptr->next = NULL;

  while (1)
    {
      /* get command string */

      if (fgets(inline,CHRLEN,in) == NULL)
        break;

      strstrip(inline);

      if (inline[0] == '!') break;
```

```
      inline[11] = NULL ; /* limit to 10 characters long */

      strcpy(cptr->string,inline);

      /* get command name */

      if (fgets(inline,CHRLEN,in) == NULL)
        break;

      strstrip(inline);

      strcpy(cptr->name,inline);

      /* get command exectution string */

      if (fgets(inline,CHRLEN,in) == NULL)
        break;

      strstrip(inline);

      strcpy(cptr->command,inline);

      /* allocate new CMD structure */

      cptr->next = (CMD *) calloc((unsigned) 1 , sizeof(CMD));

      cptr = cptr->next;
      cptr->next = NULL;
    }

  strcpy(cptr->string,"q");
  strcpy(cptr->name,"Quit");
  strcpy(cptr->command,"exit()");

  strcpy(last_command,cmd.string);
  strcpy(last_all,cmd.string);

  inline[0] = NULL;

  for (i = 1 ; i < argc ; i++)
    {
      strcat(inline,argv[i]);
      strcat(inline," ");
    }
  flag = (inline[0] != NULL) ? 1 : 0;

  while (1)
    {
      if (flag == 0)
        {
          printf("\n  %s   Options:\n\n",comm);

          for (cptr = &cmd ; cptr != NULL ; cptr = cptr->next)
```

```
        printf("   %10s :: %s\n", cptr->string, cptr->name);
      printf("  Enter Option : ");
      gets(inline);
      strstrip(inline);

      /* see if should repeat last command */

      if (strcmp(inline,"!!") == 0)
        strcpy(inline,last_all);
      else
        strcpy(last_all,inline);
    }

  /* strip off command from arguments */

  for (i = 0 ; inline[i] != NULL && inline[i] != ' ' &&
       inline[i] != '\t' ; i++)
    {
      line[i] = inline[i];
      inline[i] = ' ';
    }
  line[i] = NULL;
  strstrip(inline);

  /* see if its an exclamation marker , if so, copy last command */

  if (strcmp(line,"!") == 0 || strcmp(line,"!!") == 0)
    strcpy(line,last_command);

  /* find the proper command */

  for (cptr = &cmd ; cptr != NULL ; cptr = cptr->next)
    if (strcmp(cptr->string,line) == 0) break;

  /* continue if didn't find the command */

  if (cptr == NULL)
    {
      flag = 0;
      printf("\n *** Command Not Found\n");
      continue;
    }

  /* quit if cptr->next is NULL */

  if (cptr->next == NULL)
    exit();

  /* manufacture the system call */

  strcpy(outline,cptr->command);
  strcat(outline," ");
  strcat(outline,inline);
```

```
      /* update last_command  */

      strcpy(last_command,cptr->string);

      /* print message if flag = 1 */

      if (flag == 1) printf("\n Executing : %s\n\n",outline);

      /* execute the system call */

      system(outline);

      /* see if flag is set */

      if (flag == 1) exit();

    }

}



/* strstrip */

/* strstrip strips a string of leading and trailing spaces and tabs */

strstrip(s)
char *s;
{
  int i,j;

  /* find first non space or tab */

  for (i = 0 ; s[i] == ' ' || s[i] == '\t' ; i++);

  /* copy string */

  for (j = 0 ; s[i] != NULL ; s[j++] = s[i++]);

  s[j] = NULL;

  /* delete trailing spaces and tabs and Cr*/

  for (j = strlen(s) - 1 ; s[j] == ' ' || s[j] == '\t' ||
       s[j] == '\n' ; s[j--] = NULL);
}
```

D.2 sepsip_util.menu

The executable code for the menu driver listed in the previous section is in the file **sepsip_util**. The menu text must therefore be located in the file **sepsip_util.menu** which is listed here:

```
!
e
     Editor      ->  emacs
     /mit/13.411/sepsip/emacs_
p
     Plotting    ->  Norplot
     /mit/13.411/menu/_xterm /mit/13.411/norplot/Norplot
?
     List Directory
     ls -al
*
     Execute System Command
     /mit/13.411/menu/sys
+
     Screendump to Default Printer
     xwd | xpr -device ps | lpr
!
```

APPENDIX E

PORTABILITY CONSIDERATIONS

In the development of SEPSIP, portability considerations played a major influence. A number of SEPSIP's attributes and its structure are a direct result of the desire to be able to easily transport the source code to other computer systems. There are however, several files that may need to be edited for proper operation on other systems:

file_options.c

The function **change_directory()** may require modification on systems not using the UNIX operating system. The routine presently makes one system call to **pwd** to list the present working directory. Furthermore, the system library function **chdir()** may not be available on all systems. Eliminating the contents of this routine will only affect the change directory option of the

sepsip.c

The beginning the file **sepsip.c** contains two define directives that are system dependent. These statements and their present assignment are:

```
#define DIR "ls -al"
#define CMD "/mit/13.411/sepsip/sepsip_util"
```

DIR is a string containing the system command for listing the current directory. CMD is the path and name of the menu driver for the utility option.

APPENDIX F

SEPSIP SOURCE CODE

```
SEPSIP.CONFIGUR

Norbert H. Doerry
27 March 1989


This is the latest configuration of SEPSIP as of 19 April 1989

Version 1.0 of 27 March 1989


check_name.c              1 March 1989
        check_name()

dump_data.c              13 March 1989
        dump_data()

dump_device.c            15 March 1989
        dump_device()

edit_simulate.c          10 April 1989
        edit_simulate()
        edit_time()
        edit_jacob()
        edit_display()
        delete_pv()
        add_pv()
        edit_ref()
        split_elm()
        split_ref()

elm_jacob.c               6 March 1989
        elm_jacob()

file_options.c           19 April 1989
        file_options()
        write_sim()
        read_sim()
        write_init()
        load_init()
        change_directory()  *** this routine is system dependent ***

gauss_eliminate.c         14 February 1989
        gauss_eliminate()

integ.c                   18 October 1988
        integ()

ioliba.c                  25 March 1989
        stofa()
        stoda().
```

```
        Stofa()
        Stoda()
        getflta()
        fgetflta()
        parse()
        Parse()
        suctolc()
        slctouc()
        strcmpa()
        strncmpa()
        strsplit()
        strstrip()
        strextract()

        *** See file "ioliba.help" for details ***

load_device.c             22 October 1988
        load_device()


load_element.c             9 January 1989
        load_element()

load_initial.c            20 January 1989
        load_initial()
        open_include()
        read_init()
        read_ext_init()
        read_node_volt()

load_network.c            26 January 1989
        load_network()

load_simulation.c         27 January 1989
        load_simulation()
        read_value()
        read_display()
        read_external()
        read_reference()
        set_defaults()

make_jacobian.c                  15 February 1989
        make_jacob()

print_network.c           18 January 1989
        print_network()
        line_counter()

read_device.c              6 March 1989
        read_device()

read_element.c            25 October 1988
        read_element()
```

```
read_network.c              17 April 1989
        read_network()
        read_sub_node()
        count_char()
        make_str()
        fgets_multiple()

sepsip.c                    10 April 1989
        main()          *** This routine is system dependent ***
        load_file()
        get_filename() *** This routine is system dependent ***
        display_data()
        utilities()

setup_simulation.c          28 February 1989
        setup_simulation()


simulate.c                  14 March 1989
        **** Revision a:    29 March 1989

        run_simulation()
        initialize_simulation()
        calc_implicit()
        implicit_error()
        make_implicit()
        check_queue()
        update_variables()
        print_output()
        print_matrix()
```

```
/* check_name.c */
/* Norbert H. Doerry

   1 March 1989

*/
/* This routine checks all the element names and node names  to ensure that
   they are not multiply defined.  It then checks the subnode names for each
   node and ensures they aren't multiply defined
*/
#include <stdio.h>
#include <math.h>
#include "doerry.h"

check_name(nn,nnode,ee,nelm,errflag)
NODE **nn;
int nnode;
ELEMENT **ee;
int nelm;
int *errflag;
{
  int i,j,k;

  /* check the element names */

  for (i = 0 ; i < nelm ; i++)
    {

      /* see if another element has the same name */

      for (j = i + 1 ; j < nelm ; j++)
        {
          if (strcmp(ee[i]->name,ee[j]->name) == 0)
            {
              printf(" *** ERROR : ELEMENT %s multiply defined\n",ee[i]->name);
              *errflag = 1;
            }
        }

      /* see if a node has the same name */

      for (j = 0 ; j < nnode ; j++)
        {
          if (strcmp(ee[i]->name,nn[j]->name) == 0)
            {
              printf(" *** ERROR : %s defined as both ELEMENT and NODE\n",
                     ee[i]->name);
              *errflag = 1;
            }
        }

    }

  /* check the node names */
```

```
  for (i = 0 ; i < nnode ; i++)
    {

      /* see if node is multiply defined */

      for (j = i + 1 ; j < nnode ; j++)
        {
          if (strcmp(nn[i]->name,nn[j]->name) == 0)
            {
              printf(" *** ERROR : NODE %s multiply defined\n",nn[i]->name);
              *errflag = 1;
            }
        }

      /* check the subnodes out */

      for (j = 0 ; j < nn[i]->nbr_subnode ; j++)
        {
          for (k = j + 1 ; k < nn[i]->nbr_subnode ; k++)
            {
              if (strcmp(nn[i]->subnode[j]->name,nn[i]->subnode[k]->name) == 0)
                {
                  printf(" ERROR : SUBNODE %s : %s multiply defined\n",
                         nn[i]->name,nn[i]->subnode[j]->name);
                  *errflag = 1;
                }
            }
        }
    }
}
```

```
/* commands.c */
/* 29 November 1988

   Norbert H. Doerry

   This file contains the code for the following commands of SEPSIP:

         device_summary
         display_device
         element_summary
         display_element

*/

#include <stdio.h>
#include <math.h>
#include "doerry.h"


/* Device Summary :

   This routine presents a list of all the available devices.  If the
   output stream is stdout, the user is prompted to hit the return
   key after a page has been printed out.  A 'q' will terminate the
   listing.
*/

device_summary(dev,ndev,out)
DEVICE **dev;
int ndev;
FILE *out;
{
  int i,page,lasti,flag;
  char inline[MAXCHAR];

  fprintf(out,"\n\n               ---  DEVICE SUMMARY ---\n");

  for (i = lasti = flag = 0,page = 0 ; i < ndev  ; i++)
    {

      /* see if should start a new page */

      if ( i % (LINES_PER_PAGE - 4) == 0 && i != ndev - 1)
        {
          page++;
          if (flag != 0 && out == stdout)
            {
              printf("\n            Enter <RETURN> to continue : ");
              gets(inline);
              strstrip(inline);
              if (inline[0] == 'q' || inline[0] == 'Q') return;
              if (inline[0] == 'b' || inline[0] == 'B')
                {
                  i = lasti - LINES_PER_PAGE + 4;
```

```
                if (i < 0) i = 0;
                page -= 2;
                if (page < 1) page = 1;
                lasti -= LINES_PER_PAGE - 4;
                if (lasti < 0) lasti = 0;
              }
            else
              lasti = i;

          }
        flag = 1;  /* used to prevent starting of new page */

        fprintf(out,"\n\n             ---  DEVICE NAME ---    (PAGE %d)\n\n",
                page);
      }

    /* print out the element names */

    fprintf(out,"           %s\n",dev[i]->name);

    /* see if at end of listing */

    if ( i == ndev - 1 && out == stdout)
      {
        printf("\n SUMMARY COMPLETE : Enter <RETURN> to continue : ");
        gets(inline);

        if (inline[0] == 'b' || inline[0] == 'B')
          {
            flag = 0;
            i = lasti - LINES_PER_PAGE + 3;
            if (i < 0) i = -1 ;
            page -= 2;
            if (page < 0) page = 0;
            lasti -= LINES_PER_PAGE - 4;
            if (lasti < 0) lasti = 0;
          }
      }

  }

}


display_device(dev,ndev,inline)
DEVICE **dev;
int ndev;
char *inline;
{
  char inlin[MAXCHAR];
  int i;

  /* pull off device name if it is there */
```

```
  strstrip(inline);
  inline[0] = ' ';
  strstrip(inline);

  if (inline[0] == NULL)
    {

      /* prompt user for device name if not specified on command line */

      printf("\n\n Enter DEVICE NAME : ");
      gets(inlin);
      strstrip(inlin);
      if (inlin[0] == NULL) return;
    }
  else
    strcpy(inlin,inline);


  for (i = 0 ; i < ndev && strcmp(dev[i]->name,inlin) != 0 ; i++);

  if (i == ndev)
    {
      printf("\n\n ***ERROR : %s does not exist\n\n",inlin);
      return;
    }

  print_device(dev[i],stdout);

  printf("\n DEVICE DISPLAY COMPLETE : Enter <RETURN> to continue : ");
  gets(inlin);


}

element_summary(e,nelm,out,line)
ELEMENT **e;
int nelm;
FILE *out;
char *line;
{
  int i,j,lastj[20],page,lasti,flag;
  char inline[MAXCHAR];


  /* strip off first character of line and any following spaces or tabs */
  strstrip(line);
  line[0] = ' ';
  strstrip(line);

  fprintf(out,"\n\n               ---  ELEMENT SUMMARY ---\n");

  j = 0;  /* pointer to first element */
  lastj[0] = 0;
  page = 0;
```

```
  flag = 0;

  for (i = 0 ; i < nelm ; i++)
    {
      if ((j == (LINES_PER_PAGE - 4) || i == 0) && i != nelm - 1)
        {
          j = 0;
          page++;
          if (flag != 0 && out == stdout)
            {

              printf("\n            Enter <RETURN> to continue : ");
              gets(inline);
              strstrip(inline);
              if (inline[0] == 'q' || inline[0] == 'Q') return;
              if (inline[0] == 'b' || inline[0] == 'B')
                {
                  page = (page <= 2) ? 1 : page -= 2;
                  i = lastj[page - 1];
                }
              else
                {
                  lastj[page - 1] = i;
                }
            }
          flag = 1; /* prevents prompting for return for first page */

          fprintf(out,"\n\n            ---  ELEMENT NAME ---    (PAGE %d)\n\n",
                  page);
        }

      /* print out the element names */

      if (line[0] == 'a' || e[nelm - 1 - i]->flag == 1)
        {
          fprintf(out," %20s  ||  %-30s",e[nelm - 1 - i]->name,
                  e[nelm - 1 - i]->device->name);

          if (e[nelm - 1 - i]->flag == 0)
            fprintf(out,"  *** Not Used ***\n");
          else
            fprintf(out,"\n");

          j++;
        }

      /* see if at end of listing */

      if ( i == nelm - 1 && out == stdout)
        {
          printf("\n SUMMARY COMPLETE : Enter <RETURN> to continue : ");
          gets(inline);

          if (inline[0] == 'b' || inline[0] == 'B')
```

```
            {
              flag = 0;
              page = (page <= 2) ? 0 : page -= 2;
              i = lastj[page] -  1;
              j = LINES_PER_PAGE - 4;

            }
        }

    }

}

display_element(e,nelm,inline,q,nq)
ELEMENT **e;
int nelm;
char *inline;
QUEUE **q;
int nq;
{
  char inlin[MAXCHAR];
  int i;

  /* pull off element name if it is there */

  strstrip(inline);
  inline[0] = ' ';
  strstrip(inline);

  if (inline[0] == NULL)
    {

      /* prompt user for element name if not specified on command line */

      printf("\n\n Enter ELEMENT NAME : ");
      gets(inlin);
      strstrip(inlin);
      if (inlin[0] == NULL) return;
    }
  else
    strcpy(inlin,inline);


  for (i = 0 ; i < nelm && strcmp(e[i]->name,inlin) != 0 ; i++);

  if (i == nelm)
    {
      printf("\n\n ***ERROR : %s does not exist\n\n",inlin);
      return;
    }

  print_element(e[i],stdout,q,nq,i);

}
```

```
print_element(e,out,q,nq,eptr)
ELEMENT *e;
FILE *out;
QUEUE **q;
int nq;
int eptr;  /* pointer in element array */
{
  char inline[MAXCHAR];
  int i,j,k,l;

  fprintf(out,"\n\n Element : %-20s <>  Device : %-20s\n\n",
          e->name,e->device->name);
  fprintf(out," Element Parameters : \n\n");

  for (j = 0,i = 0 ; j < e->device->nbr_param ; i++,j++)
    {
      fprintf(out," %20s : %10.5g\n",
              e->device->param_name[j] , e->con.param[j]);


      /* after (LINES_PER_PAGE - 4) lines, wait for user to hit return
         before continuing */

      if (i% (LINES_PER_PAGE - 4) == 0 && i != 0 && out == stdout)
        {
          printf(" Hit <Return> to continue : ");
          gets(inline);
          strstrip(inline);
          if (inline[0] == 'q' || inline[0] == 'Q') return;
          if (inline[0] == 'b' || inline[0] == 'B')
            {
              j -=  2 * (LINES_PER_PAGE - 4);
              if (j < 0) j = 0;
            }
        }

    }

  if (e->con.nbr_ext_out > 0)
    {
      fprintf(out,"\n External Output Variables : \n\n");
      i++;

      for (j = 0 ; j < e->con.nbr_ext_out ; j++,i++)
        {
          fprintf(out," %20s : %12g\n", e->device->ext_out_name[j],
                  e->con.ext_out[j]);
/*          if (e->con.switch_ext_out[j] == 0)
            fprintf(out,"OFF\n");
          else
            fprintf(out,"ON\n"); */
```

```
        /* after (LINES_PER_PAGE - 4) lines, wait for user to hit return
           before continuing */

        if (i% (LINES_PER_PAGE - 4) == 0 && i != 0 && out == stdout)
          {
            printf(" Hit <Return> to continue : ");
            gets(inline);
            strstrip(inline);
            if (inline[0] == 'q' || inline[0] == 'Q') return;
            if (inline[0] == 'b' || inline[0] == 'B')
              {
                j -=  2 * (LINES_PER_PAGE - 4);
            if (j < 0) j = 0;
              }
          }

      }
  }

if (e->con.nbr_ext_in > 0)
  {
    fprintf(out,"\n External Input Variables : \n\n");
    fprintf(out,"   Time   ::       Variable       : Value\n");
    i++;

    /* print out initial values of all the External Input Variables */

    for (j = 0 ; j < e->con.nbr_ext_in ; j++,i++)
      {
        fprintf(out," %8.3f :: %-20s : %f\n", 0.0 ,
                e->device->ext_in_name[j] ,
                e->con.init_ext_in[j]);

        /* after (LINES_PER_PAGE - 4) lines, wait for user to hit return
           before continuing */

        if (i% (LINES_PER_PAGE - 4) == 0 && i != 0 && out == stdout)
          {
            printf(" Hit <Return> to continue : ");
            gets(inline);
            strstrip(inline);
            if (inline[0] == 'q' || inline[0] == 'Q') return;
            if (inline[0] == 'b' || inline[0] == 'B')
              {
                j -=  2 * (LINES_PER_PAGE - 4);
                if (j < 0) j = 0;
              }
          }

      }
```

```
    /* print out entries for this element in QUEUE array */

    for (j = 0 ; j < nq ; j++)
      {
        if (q[j]->elm != eptr)
          continue;  /* not this element */

        fprintf(out," %8.3f :: %-20s : %f\n", q[j]->time,
                e->device->ext_in_name[q[j]->var] , q[j]->value);

        /* after (LINES_PER_PAGE - 4) lines, wait for user to hit return
           before continuing */

        i++;

        if (i% (LINES_PER_PAGE - 4) == 0 && i != 0 && out == stdout)
          {
            printf(" Hit <Return> to continue : ");
            gets(inline);
            strstrip(inline);
            if (inline[0] == 'q' || inline[0] == 'Q') return;
            if (inline[0] == 'b' || inline[0] == 'B')
              {
                j -=  2 * (LINES_PER_PAGE - 4);
                if (j < 0) j = 0;
              }
          }

      }

  }

if (e->con.nbr_inputs > 0)
  {
    fprintf(out,"\n Input Variable Initial and Present Values : \n\n");
    i++;

    for (j = 0 ; j < e->con.nbr_inputs ; j++,i++)
      {
        fprintf(out," %20s : %12g : %12g\n", e->device->input_name[j],
                e->con.init_in[j],e->con.in[j]);

        /* after (LINES_PER_PAGE - 4) lines, wait for user to hit return
           before continuing */

        if (i% (LINES_PER_PAGE - 4) == 0 && i != 0 && out == stdout)
          {
            printf(" Hit <Return> to continue : ");
            gets(inline);
            strstrip(inline);
            if (inline[0] == 'q' || inline[0] == 'Q') return;
            if (inline[0] == 'b' || inline[0] == 'B')
              {
                j -= 2 * (LINES_PER_PAGE - 4);
```

```
                if (j < 0) j = 0;
              }
            }

        }
    }

  if (e->con.nbr_states > 0)
    {
      fprintf(out,"\n State Variable Initial and Present Values : \n\n");
      i++;

      for (j = 0 ; j < e->con.nbr_states ; j++,i++)
        {
          fprintf(out," %20s : %12g : %12g\n", e->device->state_name[j],
                  e->con.init_state[j],e->con.state[j]);

          /* after (LINES_PER_PAGE - 4) lines, wait for user to hit return
             before continuing */

          if (i% (LINES_PER_PAGE - 4) == 0 && i != 0 && out == stdout)
            {
              printf(" Hit <Return> to continue : ");
              gets(inline);
              strstrip(inline);
              if (inline[0] == 'q' || inline[0] == 'Q') return;
              if (inline[0] == 'b' || inline[0] == 'B')
                {
                  j -=  2 * (LINES_PER_PAGE - 4);
                  if (j < 0) j = 0;
                }
            }

        }
    }

  if (e->con.nbr_implicit > 0)
    {
      fprintf(out,"\n Implicit Variable Present Values : \n\n");
      i++;

      for (j = 0 ; j < e->con.nbr_implicit ; j++,i++)
        {
          fprintf(out," %20s : %12g\n", e->device->implicit_name[j],
                  e->con.implicit[j]);

          /* after (LINES_PER_PAGE - 4) lines, wait for user to hit return
             before continuing */

          if (i% (LINES_PER_PAGE - 4) == 0 && i != 0 && out == stdout)
            {
              printf(" Hit <Return> to continue : ");
              gets(inline);
              strstrip(inline);
```

```
            if (inline[0] == 'q' || inline[0] == 'Q') return;
            if (inline[0] == 'b' || inline[0] == 'B')
              {
                j -=  2 * (LINES_PER_PAGE - 4);
                if (j < 0) j = 0;
              }
          }

      }
  }

if (e->con.nbr_inputs > 0 && e->con.nbr_implicit > 0)
  {
    fprintf(out,"\n Jacobian Matrix Present Values : \n\n");
    i++;

    for (l = 0 ; l * 5 < e->con.nbr_inputs ; l++)
      {
        for (j = 0 ; j < e->con.nbr_implicit ; j++ , i++)
          {
            for (k = 5 * l ; k < e->con.nbr_inputs && k < (l + 1) * 5 ; k++)
              printf(" %15g",e->con.jacob_in[j + e->con.nbr_implicit * k]);
            printf("\n");

            /* after (LINES_PER_PAGE - 4) lines, wait for user to hit return
               before continuing */

            if (i% (LINES_PER_PAGE - 4) == 0 && i != 0 && out == stdout)
              {
                printf(" Hit <Return> to continue : ");
                gets(inline);
                strstrip(inline);
                if (inline[0] == 'q' || inline[0] == 'Q') return;
                if (inline[0] == 'b' || inline[0] == 'B')
                  {
                    j -=  2 * (LINES_PER_PAGE - 4);
                    if (j < 0) j = 0;
                  }
              }
          }
        printf("\n");
      }

  }


if (out == stdout)
  {
    printf("\n\n ELEMENT DESCRIPTION COMPLETE :");
    printf(" Enter <RETURN> to continue : ");
    gets(inline);
  }
fprintf(out,"\n");
}
```

```
print_device_description(k,name,param,nbr,out,types,flag)
int *k;
char *name;
char **param;
int nbr;
FILE *out;
int *types;
int flag;
{
  int i;
  char inline[MAXCHAR];
  static char *typ[] =
    {
      "Boolean",
      "Switch",
      "Integer",
      "Float"
    };

  /* see if the number of elements to print is zero */

  if (nbr <= 0) return 0;

  /* see if new page before the listing */

  if (*k + 5 > LINES_PER_PAGE && out == stdout)
    {
      *k = 0;
      printf("\n Enter <RETURN> to continue : ");
      gets(inline);
      strstrip(inline);
      if (inline[0] == 'q' || inline[0] == 'Q')
        return 1;
    }

  fprintf(out,"\n %s\n",name);

  for (i = 0 ,(*k) += 2; i < nbr; i++ ,(*k)++)
    {
      if (flag == 0 || types[i] < 0 || types[i] > 3)
        fprintf(out , "     %s\n",param[i]);
      else
        fprintf(out , "     %-7s : %s\n",typ[types[i]], param[i]);

      if (*k + 2 == LINES_PER_PAGE && out == stdout)
        {
          *k = 0;
          printf("\n Enter <RETURN> to continue : ");
          gets(inline);
          strstrip(inline);
          if (inline[0] == 'q' || inline[0] == 'Q')
            return 1;
        }
```

```
    }
  return 0;
}

print_device(d,out)
DEVICE *d;
FILE *out;
{
  int i,k;
  char inline[MAXCHAR];


  fprintf(out,"\n Device %3d : %s\n",d->type,d->name);

  k = 2;

  if (print_device_description(&k," Input Variables",
                          d->input_name,d->nbr_inputs,out,&i,0) == 1)
    return;

  if (print_device_description(&k," State Variables",
                          d->state_name,d->nbr_states,out,&i,0) == 1)
    return;

  if (print_device_description(&k," Implicit Variables",
                          d->implicit_name,d->nbr_implicit,out,&i,0) == 1)
    return;

  if (print_device_description(&k," External Input Variables",
                          d->ext_in_name,d->nbr_ext_in,out,
                          d->type_ext_in,1) == 1)
    return;

  if (print_device_description(&k," External Output Variables",
                          d->ext_out_name,d->nbr_ext_out,out,
                          d->type_ext_out,1) == 1)
    return;

  if (print_device_description(&k," Parameters",
                          d->param_name,d->nbr_param,out,&i,0) == 1)
    return;

}
```

```
/* doerry.h */
/* Norbert H. Doerry
   28 Sept 88
   modified 9 Jan 89
   modified 15 Feb 89

   This include file is designed to test the configuration of interface
   structures for my thesis.

   The variables are divided into the following categories:

   input variables    : variables that are connected to system nodes.  These
                        variables are all implicitely defined.

   state variables    : variables that represent the internal state of the
                        devices.  This is not a 'state' in the strict sense
                        of the term.  It is instead a variable whose value
                        must be 'remembered' to determine the next state
                        of the device. The 'old_state' variables are the
                        value of the state variables during the last
                        time step.

   implicit variables : variables that are driven to zero in the Newton
                         Raphson method.

   external inputs    : variables that the operator will be allowed to change
                        directly during the execution of the simulatiion.
                        These inputs can be of the following types :

                          Boolean   ( True or False)
                          Switch    ( On or Off)
                          Integer
                          Floating Point

   external output    : variables that the operator can select to display during
                        the simulation or have saved to a file for plotting


   parameters         : These are the specific parameters for a device
                        which may be different for different elements

   There are also several other definitions that must be made

      DEVICE  : is a model of an electrical machine, device, or node.
                (i.e. an Inducton Motor Model would be a device)

      ELEMENT : is an actual circuit element of type DEVICE.  Note that
                there can be multiple ELEMENTS of type DEVICE.
                (i.e. a 3 HP induction Motor would be an element and a
                2000 HP induction Motor would be another element of the
                same type.)
```

```
doerry.h


*/



#define MAXCHAR 81
#define PI 3.1415926536
#define LINES_PER_PAGE 25


/* define types */

#define BOOLEAN 0
#define SWITCH  1
#define INTEGER 2
#define FLOAT   3

typedef struct Connect
{
  int nbr_inputs;      /* number of input variables                        */
  int nbr_states;      /* number of internal states                        */
  int nbr_implicit;    /* number of implicit equations                     */
  int nbr_ext_in;      /* number of external input variables               */
  int nbr_ext_out;     /* number of external output variables              */
  int nbr_param;       /* number of parameters                             */
  double *in;          /* pointer to array of input variables              */
  double *state;       /* pointer to array of state variables              */
  double *old_state;   /* pointer to array of 'old' state variables        */
  double *implicit;    /* pointer to array of implicit variabales          */
  double *ext_in;      /* pointer to array of external input variables     */
  double *ext_out;     /* pointer to array of external output variables    */
  double *param;       /* pointer to array of parameters                   */
  double *init_state; /* pointer to array of initial values for states     */
  double *init_ext_in;/* pointer to array of initial values for ext_in     */
  double *init_in;     /* pointer to array of initial values for inputs    */
  double *jacob_in;    /* pointer to jacobian matrix of implicit variables
                          with respect to input variables                  */
  int jacob_switch;    /* = 1 if jacobian calculated by  function
                          = 0 if jacobian not calculated by function       */
  int *type_ext_in;    /* pointer to array of external input types         */
  int *type_ext_out;   /* pointer to array of external output types        */
  int *imp_index;      /* pointer to array of indexes for itab array       */
}
CONNECT;

typedef struct Element
{
  int serial;               /* serial number of element                  */
  char *name;               /* pointer to name of element                */
  struct Connect con;       /* connection pointers and counters          */
  struct Device *device;    /* pointer to device                         */
  int flag;                 /* if flag = 1, element is used in Network
                               if flag = 0, element is not used          */
}
ELEMENT;
```

```
typedef struct Device
{
  int type;             /* type of device code                          */
  int (*f)();           /* starting address of routine for this type    */
  char *name;           /* pointer to name of device                    */
  int nbr_inputs;       /* number of input variables (default)          */
  int nbr_states;       /* number of internal states (default)          */
  int nbr_implicit;     /* number of implicit equations                 */
  int nbr_ext_in;       /* number of external input variables           */
  int nbr_ext_out;      /* number of external output variables          */
  int nbr_param;        /* number of parameters                         */
  char **input_name;    /* pointer to an array of strings holding the
                           names of the input variables                 */
  char **state_name;    /* pointer to an array of strings holding the
                           names of the states                          */
  char **implicit_name;/* pointer to an array of strings holding the
                           names of the implicit equations              */
  char **ext_in_name;   /* pointer to an array of strings holding the
                           names of the external input variables        */
  char **ext_out_name; /* pointer to an array of strings holding the
                           names of the external output variables       */
  char **param_name;    /* pointer to an array of strings holding the
                           names of the paramaters                       */
  int *type_ext_in;     /* pointer to array of external input types      */
  int *type_ext_out;    /* pointer to array of external output types     */
}
DEVICE;

typedef struct Node
{
  char *name;                 /* name of node                          */
  int nbr_subnode;            /* number of subnodes                    */
  struct Subnode **subnode;   /* array of pointers to subnodes         */
  struct Node *last;          /* pointer to last Node structure        */
}
NODE;

typedef struct Subnode
{
  int type;                   /* type of subnode = 0 for voltage law
                                                 = 1 for current law     */
  int ref_flag;               /* ref subnode flag =0 not ref : =1 is ref  */
  double init_volt;           /* initial value of voltage if required     */
  char *name;                 /* name of subnode                          */
  int nbr_connect;            /* number of connections at subnode         */
  char **element;             /* array of element names                   */
  char **variable;            /* array of variable names                  */
  int *elm_ptr;               /* pointer within element array of elements */
  int *var_ptr;               /* pointer within input array               */
  struct Subnode *last;        /* pointer to last Subnode Structure       */
}
SUBNODE;
```

```
typedef struct Stream_Ptr
{
  FILE *in;                      /* present stream              */
  struct Stream_Ptr  *last;      /* last stream                 */
  char filename[MAXCHAR];        /* filename of stream          */
  int line_nbr;                  /* current line number in file */
}
STREAM_PTR;

typedef struct Queue
{
  int elm;                    /* pointer for which element in the array    */
  int var;                    /* pointer for which external input variable */
  double value;               /* new value                                 */
  double time;                /* time to take on new value                 */
  struct Queue *last;         /* pointer to last Queue structure           */
}
QUEUE;

typedef struct Simulate
{
  double dt;                 /* time increment in seconds               */
  double tmin;               /* starting time of simulation             */
  double tmax;               /* maximum time of simulation              */
  double time;               /* present time of simulation              */
  int max_iteration;         /* maximum number of iterations in NR      */
  double converge;           /* convergence limit for NR                */
  double delta;              /* percent change of variable for jacobian */
  double delta_min;          /* minimum change of variable for jacobian */
  double print_dt;           /* time increment for printing variables   */
}
SIMULATE;

typedef struct Xtable
{
  int nbr;                 /* number of variables tied to this variable */
  int *e;                  /* array of element indexes                  */
  int *v;                  /* array of input variable indexes           */
  int *mult;               /* array of multipliers for variables        */
}
XTABLE;

typedef struct Itable
{
  int e;                   /* index to element array          */
  int i;                   /* index to implicit variable array */
}
ITABLE;

typedef struct Print_var
{
  int e;                  /* index to element or node array             */
  int v;                  /* index to variable or subnode array         */
  int typ;                /* type of variable  = 0 for external output
```

```
                                        = 1 for external input
                                        = 2 for Node Voltage     */
  struct Print_var *next; /* pointer to next PRINT_VAR structure        */
}
PRINT_VAR;
```

```
/* dump_data.c */
/* Norbert H. Doerry

   13 March 1989

   This routine dumps the current state of the simulation to a file or
   the screen
*/

#include <stdio.h>
#include <math.h>
#include "doerry.h"

dump_data(out,ee,nelm,nn,nnode,qq,nq,simulate,pv,xtab,nxtab,itab,nitab)
FILE *out;
ELEMENT **ee;
int nelm;
NODE **nn;
int nnode;
QUEUE **qq;
int nq;
SIMULATE simulate;
PRINT_VAR *pv;
XTABLE **xtab;
int nxtab;
ITABLE **itab;
int nitab;
{
  int i,j,k,l;
  double *jacob,*implicit,*var,*imp;
  char *calloc();
  PRINT_VAR *temp;
  double implicit_error();

  /* print out element data */

  for (i = 0 ; i < nelm ; i++)
    {
      /* print out header */

      fprintf(out," ELEMENT : %-20s <> SERIAL : %3d <> DEVICE : %-20s ",
              ee[i]->name,ee[i]->serial,ee[i]->device->name);
      if (ee[i]->flag) fprintf(out,"\n");
      else fprintf(out,"<> *** UNUSED ***\n");

      /* print out the parameters */

      if (ee[i]->con.nbr_param > 0)
        {
          fprintf(out,"\n PARAMETER Values\n");

          for (j = 0 ; j < ee[i]->con.nbr_param ; j++)
            {
              fprintf(out,"  %20s :: %15g\n",
```

```
                    ee[i]->device->param_name[j],
                    ee[i]->con.param[j]);
            }
        }

    /* print out input variables */

    if (ee[i]->con.nbr_inputs > 0)
        {
        fprintf(out,"\n INPUT Variable Initial and Present Values\n");

        for (j = 0 ; j < ee[i]->con.nbr_inputs ; j++)
            {
            fprintf(out,"  %20s :: %15g :: %15g\n",
                    ee[i]->device->input_name[j],
                    ee[i]->con.init_in[j], ee[i]->con.in[j]);
            }
        }

    /* print out the state variables */

    if (ee[i]->con.nbr_states > 0)
        {
        fprintf(out,"\n STATE Variable Initial, Old and Present Values\n");

        for (j = 0 ; j < ee[i]->con.nbr_states ; j++)
            {
            fprintf(out,"  %20s :: %15g :: %15g :: %15g\n",
                    ee[i]->device->state_name[j],
                    ee[i]->con.init_state[j],ee[i]->con.old_state[j],
                    ee[i]->con.state[j]);
            }
        }

    /* print out the implicit variables */

    if (ee[i]->con.nbr_implicit > 0)
        {
        fprintf(out,"\n IMPLICIT Variable Present Values and Index\n");

        for (j = 0 ; j < ee[i]->con.nbr_implicit ; j++)
            {
            fprintf(out,"  %20s :: %15g :: %3d\n",
                    ee[i]->device->implicit_name[j],
                    ee[i]->con.implicit[j],ee[i]->con.imp_index[j]);
            }
        }

    /* print out the external input variables */

    if (ee[i]->con.nbr_ext_in > 0)
        {
        fprintf(out,"\n EXTERNAL INPUT Initial and Present Values\n");
```

```
        for (j = 0 ; j < ee[i]->con.nbr_ext_in ; j++)
          {
            fprintf(out,"  %20s :: %15g :: %15g\n",
                    ee[i]->device->ext_in_name[j],
                    ee[i]->con.init_ext_in[j],ee[i]->con.ext_in[j]);
          }
      }

    /* print out the external output variables */

    if (ee[i]->con.nbr_ext_out > 0)
      {
        fprintf(out,"\n EXTERNAL OUTPUT Present Values\n");

        for (j = 0 ; j < ee[i]->con.nbr_ext_out ; j++)
          {
            fprintf(out,"  %20s :: %15g\n",
                    ee[i]->device->ext_out_name[j],
                    ee[i]->con.ext_out[j]);
          }
      }

    /* print out the jacobian */

    fprintf(out,"\n\n JACOBIAN SWITCH IS ");
    if (ee[i]->con.jacob_switch == 0) fprintf(out,"OFF\n");
    else fprintf(out,"ON\n");

    fprintf(out,"\n\n                    Jacobain Matrix\n\n");

    for (j = 0 ; j * 5 < ee[i]->con.nbr_inputs ; j++)
      {
        for (k = 0 ; k < ee[i]->con.nbr_implicit ; k++)
          {
            for (l = j*5 ; l <  ee[i]->con.nbr_inputs &&
                 l < (j+1)*5 ; l++)
              fprintf(out," %15g",
                      ee[i]->con.jacob_in[k + ee[i]->con.nbr_implicit * l]);

            fprintf(out,"\n");
          }
        fprintf(out,"\n");
      }

    /* print a form feed */

    fprintf(out,"\f");
  }


/* print out network information */

for (i = 0 ; i < nnode ; i++)
  {
```

```
      fprintf(out,"\n NODE :: %s\n\n",nn[i]->name);

      for (j = 0 ; j < nn[i]->nbr_subnode ; j++)
        {

          fprintf(out,"   ");

          if (nn[i]->subnode[j]->ref_flag)
            fprintf(out," REFERENCE");

          if (nn[i]->subnode[j]->type == 0)
            fprintf(out," VOLTAGE SUBNODE : ");
          else
            fprintf(out," CURRENT SUBNODE : ");

          fprintf(out,"%s ",nn[i]->subnode[j]->name );

          if (nn[i]->subnode[j]->type == 0)
            fprintf(out," :: %15-g\n",nn[i]->subnode[j]->init_volt);
          else
            fprintf(out,"\n");

          /* print out the attached variables */

          for (k = 0 ; k < nn[i]->subnode[j]->nbr_connect ; k++)
            {
              fprintf(out,"     %20s:%-20s = %15g\n",
                      nn[i]->subnode[j]->element[k],
                      nn[i]->subnode[j]->variable[k],
                      ee[nn[i]->subnode[j]->elm_ptr[k]]->
                      con.in[nn[i]->subnode[j]->var_ptr[k]]);
            }

          fprintf(out,"\n");
        }

      /* print out form feed */

      fprintf(out,"\f");
    }

  /* print out the queue */

  fprintf(out," QUEUE\n\n");

  for (i = 0 ; i < nq ; i++)
    {
      fprintf(out," %20s:%-20s = %15g at time %15g\n",
              ee[qq[i]->elm]->name,
              ee[qq[i]->elm]->device->input_name[qq[i]->var],
              qq[i]->value,
              qq[i]->time);
    }
```

```
  /* print out form feed */

  fprintf(out,"\f");

  fprintf(out," IMPLICIT VECTOR\n\n");

  fprintf(out,"     Implicit Error = %-15g\n\n",implicit_error(ee,nelm));

  for (i = 0 ; i < nitab ; i++)
    {
      fprintf(out,"  %2d  %20s:%-20s = %15g\n",i,
              ee[itab[i]->e]->name,
              ee[itab[i]->e]->device->implicit_name[itab[i]->i],
              ee[itab[i]->e]->con.implicit[itab[i]->i]);
    }



  /* print out form feed */

  fprintf(out,"\f");

  fprintf(out," INPUT VARIABLE VECTOR\n\n");

  for (i = 0 ; i < nxtab ; i++)
    {
      fprintf(out," Variable %d\n",i);

      for (j = 0 ; j < xtab[i]->nbr ; j++)
        fprintf(out,"    %20s:%-20s = %14g x %3d\n",
                ee[xtab[i]->e[j]]->name,
                ee[xtab[i]->e[j]]->device->input_name[xtab[i]->v[j]],
                ee[xtab[i]->e[j]]->con.in[xtab[i]->v[j]],
                xtab[i]->mult[j]);
    }


  /* assemble and print out the jacobian matrix */

  fprintf(out,"\f");

  jacob    = (double *) calloc(nxtab * nitab , sizeof(double));
  implicit = (double *) calloc(nxtab         , sizeof(double));
  imp      = (double *) calloc(nxtab         , sizeof(double));
  var      = (double *) calloc(nxtab         , sizeof(double));

  /* make the jacobian matrix */

  make_jacob(jacob,xtab,nxtab,itab,nitab,ee,nelm,nn,nnode,simulate);

  /* print out the jacobian matrix */

  fprintf(out,"\n SYSTEM JACOBIAN MATRIX\n\n");
```

```
  for (i = 0 ; i * 5 < nxtab ; i++)
    {
      fprintf(out," %3s ","  ");
      for (k = 5 * i ; k < nxtab && k < 5 * (i+1) ; k++)
        {
          fprintf(out,"      %3d      ",k);
        }
      fprintf(out,"\n");

      for (j = 0 ; j < nitab ; j++)
        {
          fprintf(out," %3d ",j);
          for (k = 5 * i ; k < nxtab && k < 5 * (i+1) ; k++)
            {
              fprintf(out," %14g",jacob[j + nitab * k]);
            }
          fprintf(out,"\n");
        }

      fprintf(out,"\n\n");
    }

  fprintf(out,"\f");

  /* print out Newton Raphson Correction */

  /* make the implicit variable vector */

  make_implicit(ee,nelm,itab,nitab,implicit);
  make_implicit(ee,nelm,itab,nitab,imp);

  /* solve the system of equations */

  i = gauss_eliminate(nxtab,jacob,implicit,var);

  /* note that the contents of jacob and implicit were
     destroyed by gauss_eliminate */

  fprintf(out,"\n IMPLICIT VECTOR and CORRECTION VECTOR\n\n");

  if (i != 0)
    fprintf(out," *** SINGULAR SYSTEM MATRIX ***\n\n");


  fprintf(out,"\n      n      Implicit       Correction\n");
  for (i = 0 ; i < nitab ; i++)
    fprintf(out,"   %3d  %15g  %15g\n",i,imp[i],var[i]);

  fprintf(out,"\f");



  /* print out the smulation stuff */
```

```
  fprintf(out,"\n\n SIMULATION DATA\n\n");

  fprintf(out,"    DT         = %g\n",simulate.dt);
  fprintf(out,"    TMIN       = %g\n",simulate.tmin);
  fprintf(out,"    TMAX       = %g\n",simulate.tmax);
  fprintf(out,"    TIME       = %g\n",simulate.time);
  fprintf(out,"    MAX_ITER   = %d\n",simulate.max_iteration);
  fprintf(out,"    CONVERGE   = %g\n",simulate.converge);
  fprintf(out,"    DELTA      = %g\n",simulate.delta);
  fprintf(out,"    DELTA_MIN  = %g\n",simulate.delta_min);
  fprintf(out,"    PRINT_DT   = %g\n\n\n",simulate.print_dt);

  fprintf(out," DISPLAYED VARIABLES\n\n");

  for (temp = pv->next ; temp != NULL ; temp = temp->next)
    {
      if (temp->typ == 0) /* external output */
        fprintf(out,"   %20s:%-20s\n",
                ee[temp->e]->name,
                ee[temp->e]->device->ext_out_name[temp->v]);
      else if (temp->typ == 1) /* external input */
        fprintf(out,"   %20s:%-20s\n",
                ee[temp->e]->name,
                ee[temp->e]->device->ext_in_name[temp->v]);
      else /* subnode voltage */
        fprintf(out,"   %20s:%-20s\n",
                nn[temp->e]->name,
                nn[temp->e]->subnode[temp->v]->name);
    }
}
```

```
/* dump_device.c */
/* Norbert H. Doerry

   15 March 1989

   This routine prints out all the device characteristics of all the
   devices to the filename specified to it.  If the filename is not
   provided, then it is written to the stdout

*/
#include <stdio.h>
#include <math.h>
#include "doerry.h"

dump_device(dev,ndev,inline)
DEVICE **dev;
int ndev;
char *inline;
{
  int i;
  FILE *out;

  inline[0] = ' ';
  strstrip(inline);

  /* open stream */

  out = (inline[0] == NULL) ? stdout : fopen(inline,"w");
  if (out == NULL) out = stdout;

  /* print out devices */

  for (i = 0 ; i < ndev ; i++)
    {
      print_device(dev[i],out);
      if (i < ndev - 1 && out != stdout)
        fprintf(out,"\f");
      else if (out == stdout)
        {
          printf(" Enter <RETURN> to continue : ");
          gets(inline);
        }
    }

  /* close file */

  if (out != stdout) fclose(out);



}
```

```
/* edit_simulate.c */
/*  Norbert H. Doerry

    27 March  1989

    This routine allows the user to edit the simulation parameters


    *** modified 10 April 1989 to fix error in edit_ref routine ***

*/

#include <stdio.h>
#include <math.h>
#include "doerry.h"

edit_simulate(ee,nelm,nn,nnode,simulate,pv,line)
ELEMENT **ee;
int nelm;
NODE **nn;
int nnode;
SIMULATE *simulate;
PRINT_VAR *pv;
char *line;
{
  double val;
  char ch,inline[MAXCHAR],cmd;
  int ncnt,flag;


  /* strip off the edit_simulate command */
  strcpy(inline,line);
  strstrip(inline);
  inline[0] = ' ';
  strstrip(inline);

  cmd = inline[0];

  if (cmd != 't' && cmd != 'j' && cmd != 'r' && cmd != 'd')
    flag = 1;
  else
    flag = 0;

  while (1)
    {
      if (flag)
        {
          printf("\n Edit SIMULATE Section\n");
          printf("   d   Edit Display Variables\n");
          printf("   j   Edit Jacobian Parameters\n");
          printf("   q   Quit\n");
          printf("   r   Edit Reference Voltage Subnodes\n");
          printf("   t   Edit Time Parameters\n");
          printf(" Enter Command : ");
```

```
          gets(inline);
          strstrip(inline);
          cmd = inline[0];
          if (cmd == 'q') return;
          if (cmd != 't' && cmd != 'j' && cmd != 'r' && cmd != 'd')
            continue;
        }

      if (cmd == 'd')
        edit_display(inline,ee,nelm,nn,nnode,pv);
      else if (cmd == 'j')
        edit_jacob(simulate);
      else if (cmd == 'r')
        edit_ref(inline,ee,nelm,nn,nnode,simulate);
      else if (cmd == 't')
        edit_time(simulate);

      if (flag == 0) return;
    }
}

edit_time(simulate)
SIMULATE *simulate;
{
  char ch,inline[MAXCHAR];
  double val;
  int ncnt;

  /* enter time increment */

 a:
  printf(" Enter TIME_STEP      (default %12.5g) : ",simulate->dt);
  gets(inline);
  ch = Stoda(inline,&val,&ncnt,1);
  if (ch == 'q') return;
  if (ncnt == 1 && val > 0) simulate->dt = val;

 b:
  printf(" Enter TMIN           (default %12.5g) : ",simulate->tmin);
  gets(inline);
  ch = Stoda(inline,&val,&ncnt,1);
  if (ch == 'q') return;
  if (ch == 'b') goto a;
  if (ncnt == 1) simulate->tmin = val;

 c:
  printf(" Enter TMAX           (default %12.5g) : ",simulate->tmax);
  gets(inline);
  ch = Stoda(inline,&val,&ncnt,1);
  if (ch == 'q') return;
  if (ch == 'b') goto b;
  if (ncnt == 1) simulate->tmax = val;

 d:
```

```
  printf(" Enter PRINT_STEP     (default %12.5g) : ",simulate->print_dt);
  gets(inline);
  ch = Stoda(inline,&val,&ncnt,1);
  if (ch == 'q') return;
  if (ch == 'b') goto c;
  if (ncnt == 1 && val > 0) simulate->print_dt = val;

}

edit_jacob(simulate)
SIMULATE *simulate;
{
  char ch,inline[MAXCHAR];
  double val;
  int ncnt;

 e:
  printf(" Enter CONVERGE       (default %12.5g) : ",simulate->converge);
  gets(inline);
  ch = Stoda(inline,&val,&ncnt,1);
  if (ch == 'q') return;
  if (ncnt == 1 && val > 0) simulate->converge = val;

 f:
  printf(" Enter MAX_ITERATION (default %12d) : ",simulate->max_iteration);
  gets(inline);
  ch = Stoda(inline,&val,&ncnt,1);
  if (ch == 'q') return;
  if (ch == 'b') goto e;
  if (ncnt == 1 && val > 0) simulate->max_iteration = (int) val;

 g:
  printf(" Enter DELTA          (default %12.5g) : ",simulate->delta);
  gets(inline);
  ch = Stoda(inline,&val,&ncnt,1);
  if (ch == 'q') return;
  if (ch == 'b') goto f;
  if (ncnt == 1 && val > 0) simulate->delta = val;

 h:
  printf(" Enter DELTA_MIN      (default %12.5g) : ",simulate->delta_min);
  gets(inline);
  ch = Stoda(inline,&val,&ncnt,1);
  if (ch == 'q') return;
  if (ncnt == 1 && val > 0) simulate->delta_min = val;

}

edit_display(inline,ee,nelm,nn,nnode,pv)
char *inline;
ELEMENT **ee;
int nelm;
NODE **nn;
```

```
int nnode;
PRINT_VAR *pv;
{
  int i,j,typ,flag,e,v;
  PRINT_VAR *temp;
  char line[MAXCHAR],cmd,en[MAXCHAR],vn[MAXCHAR];

  /* strip inline of first character */

  strcpy(line,inline);
  line[0] = ' ';
  strstrip(line);

  cmd = line[0];

  /* see if cmd is a valid command */

  if (cmd == 'a' || cmd == 'd')
    flag = 1;
  else
    flag = 0;

  while (1)
    {
      if (flag == 0)
        {

          printf("\n                DISPLAY VARIABLES\n");

          for (temp = pv->next ; temp != NULL ; temp = temp->next)
            {
              if (temp->typ == 0)
                printf("  %20s : %-20s\n",ee[temp->e]->name,
                        ee[temp->e]->device->ext_out_name[temp->v]);
              else if (temp->typ == 1)
                printf("  %20s : %-20s\n",ee[temp->e]->name,
                        ee[temp->e]->device->ext_in_name[temp->v]);
              else if (temp->typ == 2)
                printf("  %20s : %-20s\n",nn[temp->e]->name,
                        nn[temp->e]->subnode[temp->v]->name);
            }

          if (cmd == 'q') return 0;

          printf("\n Edit DISPLAY VARIABLES COMMANDS\n");
          printf("   a  Add Display Variable\n");
          printf("   d  Delete Display Variable\n");
          printf("   q  Quit Edit Display Variables\n");
          printf(" Enter Command : ");
          gets(line);
          strstrip(line);
          cmd = line[0];
          if (cmd == 'q') return 0;
          else if (cmd != 'a' && cmd != 'd') continue;
```

```
        }

      /* strip off command */

      line[0] = ' ';
      strstrip(line);

      /* strip off element/node and variable/subnode */

      split_elm(line,en,vn);

      /* see if the element/node are there */

      if (en[0] == NULL || vn[0] == NULL)
        {
          printf("\n Enter Display Variable (Element : Variable) : ");
          gets(line);

          split_elm(line,en,vn);
        }

      /* If the entry is hosed, show the menu */

      if (en[0] == NULL || vn[0] == NULL)
        {
          flag = 0;
          continue;
        }

      /* find the element */

      for (i = 0 ; i < nelm && strcmp(ee[i]->name,en) != 0 ; i++);

      if (i == nelm)
        {
          for (i = 0 ; i < nnode && strcmp(nn[i]->name,en) != 0 ; i++);
          if (i == nnode)
            {
              printf(" *** Unable to find element/node %s\n",en);
              flag = 0;
              continue;
            }
          typ = 2;

          for (j = 0 ; j < nn[i]->nbr_subnode &&
               strcmp(nn[i]->subnode[j]->name,vn) != 0 ; j++);

          if (j == nn[i]->nbr_subnode)
            {
              printf(" *** Unable to find SUBNODE %s\n",vn);
              flag = 0;
              continue;
            }
```

```
          if (nn[i]->subnode[j]->type == 1) /* current subnode */
            {
              printf(" *** SUBNODE %s : %s is a current subnode\n",en,vn);
              flag = 0;
              continue;
            }

        }
      else /* its an element : variable entry */
        {
          /* see if its an external output variable */

          for (j = 0 ; j < ee[i]->con.nbr_ext_out &&
               strcmp(ee[i]->device->ext_out_name[j],vn) != 0 ; j++);

          if (j == ee[i]->con.nbr_ext_out) /* wasn't an external output */
            {
              for (j = 0 ; j < ee[i]->con.nbr_ext_in &&
                   strcmp(ee[i]->device->ext_in_name[j],vn) != 0 ; j++);

              if ( j == ee[i]->con.nbr_ext_in) /* wasn't an external input */
                {
                  printf(" *** Unable to find VARIABLE %s\n",vn);
                  flag = 0;
                  continue;
                }

              typ = 1;
            }
          else
            typ = 0;
        }

      /* now lets get to work */

      if (cmd == 'd')
        delete_pv(i,j,typ,pv);
      else
        add_pv(i,j,typ,pv);

      if (flag) return;

    }

}
delete_pv(i,j,typ,pv)
int i,j,typ;
PRINT_VAR *pv;
{
  PRINT_VAR *temp,*last;

  last = pv;
  for (temp = last->next ; temp != NULL ;temp = last->next )
    {
```

```
      if (temp->e == i && temp->v == j && temp->typ == typ)
        {
          last->next = temp->next;
          free(temp);
        }
      else
        {
          last = temp;
        }
    }
}

add_pv(i,j,typ,pv)
int i,j,typ;
PRINT_VAR *pv;
{
  PRINT_VAR *temp,*last;
  char *calloc();

  /* ensure that the variable isn't already there, if so, delete it */

  delete_pv(i,j,typ,pv);

  last = pv;
  for (temp = last->next ; temp != NULL ;temp = last->next )
    last = temp;

  last->next = (PRINT_VAR *) calloc((unsigned) 1 , sizeof(PRINT_VAR));
  temp = last->next;
  temp->e = i;
  temp->v = j;
  temp->typ = typ;
  temp->next = NULL;


}


edit_ref(inline,ee,nelm,nn,nnode,simulate)
char *inline;
ELEMENT **ee;
int nelm;
NODE **nn;
int nnode;
SIMULATE *simulate;
{
  int i,j,flag,ncnt;
  char line[MAXCHAR],cmd,n_name[MAXCHAR],s_name[MAXCHAR],var[MAXCHAR];
  double val;

  inline[0] = ' ';
  strstrip(inline);

  /* print out the reference voltage subnodes if the user hasn't
```

```
    entered any arguments on the command line */

  if (inline[0] == NULL)
    {
      printf("\n\n Reference Voltage Subnodes\n\n");
      for (i = 0 ; i < nnode ; i++)
        {
          for (j = 0 ; j < nn[i]->nbr_subnode ; j++)
            {
              if (nn[i]->subnode[j]->ref_flag == 0 ||
                  nn[i]->subnode[j]->type     == 1) continue;
              printf("   %20s:%-20s  %g\n",nn[i]->name,
                     nn[i]->subnode[j]->name,
                     nn[i]->subnode[j]->init_volt);
            }
        }

      printf("\n Enter Reference Voltage Subnode : ");
      gets(inline);
      strstrip(inline);
    }

  /* if still don't have anything, give up and return */

  if (inline[0] == NULL) return;

  /* decompose the string */

  split_ref(inline,n_name,s_name,var);

  /* find the node */

  for (i = 0 ; i < nnode && strcmp(n_name,nn[i]->name) != 0 ; i++);

  if (i == nnode)
    {
      printf("\n *** ERROR : Node %s Does not exist\n",n_name);
      return;
    }

  /* find the subnode */

  for (j = 0 ; j < nn[i]->nbr_subnode &&
       strcmp(s_name,nn[i]->subnode[j]->name) != 0 ; j++);

  if (j == nn[i]->nbr_subnode)
    {
      printf("\n *** ERROR : Subnode %s:%s Does not exist\n",n_name,s_name);
      return;
    }

  /* see if a new value was provided */

  Stoda(var,&val,&ncnt,1);
```

```
  if (ncnt == 0)
    {
      printf("\n Enter New Reference Voltage (default %g) : ",
             nn[i]->subnode[j]->init_volt);

      gets(var);
      strstrip(var);
      Stoda(var,&val,&ncnt,1);
      if (ncnt == 0) return;  /* give up if default desired */
    }

  /* set the initial voltage */

  nn[i]->subnode[j]->init_volt = val;


}

split_elm(in,en,vn)
char *in,*en,*vn;
{
  int i,j;

  /* strip off element/node and variable/subnode */

  for (i = 0 ; in[i] != NULL && in[i] != ':' ; i++)
    en[i] = in[i];
  en[i] = NULL;
  strstrip(en);

  if (in[i] != NULL) i++;  /* look past the colon */

  for (j = 0 ; in[i] != NULL ; i++ , j++)
    vn[j] = in[i];

  vn[j] = NULL;

  strstrip(vn);

  /* ignore everything after the first space of vn */

  for (j = 0 ; vn[j] != NULL && vn[j] != ' ' && vn[j] != '\t' ; j++);
  vn[j] = NULL;

}
split_ref(in,nn,sn,var)
char *in,*nn,*sn,*var;
{
  int i,j;

  /* strip off subnode*/

  for (i = 0 ; in[i] != NULL && in[i] != ':' ; i++)
```

```
    nn[i] = in[i];
  nn[i] = NULL;
  strstrip(nn);

  /* strip off the subnode */

  if (in[i] != NULL) i++;  /* look past the colon */

  for (j = 0 ; in[i] != NULL && in[i] != ' '  && in[i] != '\t' ; i++ , j++)
    sn[j] = in[i];
  sn[j] = NULL;
  strstrip(sn);

  /* strip off the value */

  for (j = 0 ; in[i] != NULL ; i++,j++)
    var[j] = in[i];
  var[j] = NULL;
  strstrip(var);

}
```

```
/* elm_jacob.c */
/* Norbert H. Doerry

   6 March 1989
*/
/* This routine calculates the jacobian matrix for an element by varying
   the input variables and and seeing how the implicit variables change
*/

#include <stdio.h>
#include <math.h>
#include "doerry.h"

elm_jacob(e,simulate)
ELEMENT *e;
SIMULATE simulate;
{
  int i,j,k;
  double low_in , normal_in ;
  double *in,*state,*implicit,*ext_out,*temp_implicit,*temp;
  char *calloc();

  /* ensure delta_min is large enough to prevent divide by zero errors */

  if (simulate.delta_min < 1e-20) simulate.delta_min = 1e-20;

  temp_implicit = (double *) calloc((unsigned)e->con.nbr_implicit,
                                    sizeof(double));

  state    = e->con.state;
  implicit = e->con.implicit;
  ext_out  = e->con.ext_out;

  e->con.state    = (double *) calloc((unsigned)e->con.nbr_states ,
                                      sizeof (double));
  e->con.implicit = (double *) calloc((unsigned)e->con.nbr_implicit,
                                      sizeof (double));
  e->con.ext_out  = (double *) calloc((unsigned)e->con.nbr_ext_out ,
                                      sizeof (double));


  for (i = 0 ; i < e->con.nbr_inputs ; i++)
    {
      normal_in = e->con.in[i];

      /* change the input variable to something somewhat smaller */

      e->con.in[i] = ( fabs(normal_in * simulate.delta) > simulate.delta_min) ?
        normal_in * (1.0 - simulate.delta) : normal_in - simulate.delta_min;

      low_in = e->con.in[i];

      /* calculate the new implicit variables */
```

```
      (e->device->f)(e,simulate.dt);

      /* exchange temp_implict with e->con.implicit */

      temp = e->con.implicit;
      e->con.implicit = temp_implicit;
      temp_implicit = temp;

      /* change the input variable to something somewhat larger */

      e->con.in[i] = ( fabs(normal_in * simulate.delta) > simulate.delta_min) ?
        normal_in * (1.0 + simulate.delta) : normal_in + simulate.delta_min;

      /* calculate the new implicit variables */

      (e->device->f)(e,simulate.dt);

      /* calculate the slopes */

      for (j = 0 ; j < e->con.nbr_implicit ; j++)
        {
          e->con.jacob_in[j + e->con.nbr_implicit * i] =
            (e->con.implicit[j] - temp_implicit[j])
              / (e->con.in[i] - low_in);
        }
      e->con.in[i] = normal_in;
    }

  free(e->con.state);
  free(e->con.implicit);
  free(e->con.ext_out);
  free(temp_implicit);

  e->con.state = state;
  e->con.implicit = implicit;
  e->con.ext_out = ext_out;

}
```

```
/* file_options.c */
/* Norbert H. Doerry

   13 March 1989

   This file contains the routines for saving and loading the SIMULATION
   section and the INITIALIZATION section.

   *** Modified 19 April 1989 ***

   fixed 'save simulation' to print "SIMULATION" at the top

*/
#include <stdio.h>
#include <math.h>
#include "doerry.h"

static char sim_filename[MAXCHAR],init_filename[MAXCHAR];

file_options(inline,ee,nelm,nn,nnode,q,nq,simulate,pv,
             xtab,nxtab,itab,nitab,errflag)
char *inline;
ELEMENT **ee;
int nelm;
NODE **nn;
int nnode;
QUEUE ***q;
int *nq;
SIMULATE *simulate;
PRINT_VAR *pv;
XTABLE **xtab;
int nxtab;
ITABLE **itab;
int nitab;
int errflag;
{
  int i,j,k,flag,rw;
  char line[MAXCHAR],filename[MAXCHAR],string[21],cmd;
  FILE *io;

  /* initialize flag to no errors */

  flag = 0;

  /* copy the input line */
  strcpy(line,inline);

  /* strip off the first character */

  line[0] = ' ';
  strstrip(line);

  /* get the command if its there  */
```

```
  cmd = line[0];

  /* get the filename if its there */

  if (cmd != 'NULL')
    {
      line[0] = ' ';
      strstrip(line);
      strcpy(filename,line);
    }

  /* if the command isn't a proper one, display the menu */

  flag = (cmd == NULL) ? 1 : 0;

  while (1)
    {

      if (flag)
        {
          printf("\n\n FILE OPTIONS\n");
          printf("   c   Change Working Directory\n");
          if (errflag == 0)
            {
              printf("   d   Dump Simulation State\n");
              printf("   i   Save INITIALIZATION Section\n");
              printf("   I   Load INITIALIZATION Section\n");
            }
          printf("   q   Quit\n");
          if (errflag == 0)
            {
              printf("   s   Save SIMULATION Section\n");
              printf("   S   Load SIMULATION Section\n");
            }
          printf(" Enter Command : ");

          gets(line);
          strstrip(line);
          cmd = line[0];

          /* get the filename if its there */

          if (cmd != 'NULL')
            {
              line[0] = ' ';
              strstrip(line);
              strcpy(filename,line);
            }

        }

      if (cmd == 'q')
        return flag;
```

```
    else if (cmd == 'c')
      change_directory(filename);

    else if (cmd == 'd' && errflag == 0)
      {
        if (filename[0] != NULL)
          i = get_filename(filename,&io,1,"DUMP",1);
        else
          i = get_filename(filename,&io,1,"DUMP",0);

        if (i != 0) return flag;

        dump_data(io,ee,nelm,nn,nnode,*q,*nq,*simulate,
                  pv,xtab,nxtab,itab,nitab);

        if (io != stdout) fclose(io);
      }

    else if (errflag == 0 && (cmd == 's' || cmd == 'S' ||
                              cmd == 'i' || cmd == 'I'))
      {

        if (cmd == 's' || cmd == 'i') rw = 1;   /* write to file */
        else rw = 0; /* read from file */

        if (cmd == 's' || cmd == 'S') strcpy(string,"SIMULATION");
        else strcpy(string,"INITIALIZATION");

        /* try to load file immediately if filename is non zero length */

        if (filename[0] != NULL)
          {
            i = get_filename(filename,&io,rw,string,1);
          }

        /* otherwise, prompt user for filename, and provide default name */

        else
          {
            if (cmd == 's' || cmd == 'S')
              strcpy(filename,sim_filename);
            else
              strcpy(filename,init_filename);

            i = get_filename(filename,&io,rw,string,0);
          }

        /* see if the opening of the file was aborted */

        if (i != 0)
          return flag;

        /* save the filename */
```

```
          if (cmd == 's' || cmd == 'S')
            strcpy(sim_filename, filename);
          else
            strcpy(init_filename,filename);

          /* process the files */

          if (cmd == 's')
            return write_sim(io,sim_filename,ee,nelm,nn,
                             nnode,*q,*nq,*simulate,pv);
          else if (cmd == 'S')
            return read_sim(io,sim_filename,ee,nelm,nn,
                            nnode,q,nq,simulate,pv);
          else if (cmd == 'i')
            return write_init(io,ee,nelm,nn,nnode);
          else if (cmd == 'I')
            return load_init(io,filename,ee,nelm,nn,nnode);
        }

      else flag = 1;

      if (flag == 0) return;
    }
}

write_sim(out,filename,ee,nelm,nn,nnode,q,nbrq,simulate,pv)
FILE *out;
char *filename;
ELEMENT **ee;
int nelm;
NODE **nn;
int nnode;
QUEUE **q;
int nbrq;
SIMULATE simulate;
PRINT_VAR *pv;
{
  int i,j,k;
  PRINT_VAR *temp;

  fprintf(out,"! %s\n!\n",filename);
  fprintf(out,"SIMULATION\n!\nDISPLAY\n");

  for (temp = pv->next ; temp != NULL ; temp = temp->next)
    {
      if (temp->typ == 0)
        {
          fprintf(out,"  %-20s : %s\n",ee[temp->e]->name,
                  ee[temp->e]->device->ext_out_name[temp->v]);
        }
      else if (temp->typ == 1)
        {
          fprintf(out,"  %-20s : %s\n",ee[temp->e]->name,
                  ee[temp->e]->device->ext_in_name[temp->v]);
```

```
        }
      else if (temp->typ == 2)
        {
          fprintf(out,"  %-20s : %s\n",nn[temp->e]->name,
                  nn[temp->e]->subnode[temp->v]->name);
        }
    }
  fprintf(out,"  END\n!\n");

  fprintf(out,"  TIME_STEP  %g\n",simulate.dt);
  fprintf(out,"  TMIN       %g\n",simulate.tmin);
  fprintf(out,"  TMAX       %g\n",simulate.tmax);
  fprintf(out,"  PRINT_STEP %g\n",simulate.print_dt);
  fprintf(out,"  DELTA      %g\n",simulate.delta);
  fprintf(out,"  DELTA_MIN  %g\n",simulate.delta_min);
  fprintf(out,"  CONVERGE   %g\n",simulate.converge);
  fprintf(out,"  MAX_ITERATION %d\n",simulate.max_iteration);
  fprintf(out,"  REFERENCE\n");

  for (i = 0 ; i < nnode ; i++)
    for (j = 0 ; j < nn[i]->nbr_subnode ; j++)
      {
        if (nn[i]->subnode[j]->ref_flag == 0)
          continue;
        if (nn[i]->subnode[j]->type == 0)
          fprintf(out,"    V : %20s : %-20s %g\n",nn[i]->name ,
                  nn[i]->subnode[j]->name , nn[i]->subnode[j]->init_volt);
        else
          fprintf(out,"    I : %20s : %-20s\n",nn[i]->name ,
                  nn[i]->subnode[j]->name);
      }
  fprintf(out,"    END\n!\n");

  fprintf(out,"EXTERNAL INPUTS\n");

  for (i = 0 ; i < nbrq ; i++)
    {
      fprintf(out,"  %20s : %20s  %10g  %10g\n", ee[q[i]->elm]->name,
              ee[q[i]->elm]->device->ext_in_name[q[i]->var] , q[i]->value,
              q[i]->time);
    }

  fprintf(out,"  END\n!\n");

  if (out != stdout) fclose(out);

  return 0;

}

read_sim(in,filename,ee,nelm,nn,nnode,q,nbrq,simulate,pv)
FILE *in;
char *filename;
ELEMENT **ee;
```

```
int nelm;
NODE **nn;
int nnode;
QUEUE ***q;
int *nbrq;
SIMULATE *simulate;
PRINT_VAR *pv;
{
  STREAM_PTR *strm,stm;
  int errflag,i,j;
  PRINT_VAR ppv,*temp_pv,*old;
  QUEUE **qq;
  SIMULATE ss;
  int nq;


  /* initialize stream pointer structure */

  strm = &stm;

  stm.in = in;
  strcpy(stm.filename , filename);
  stm.line_nbr = 0;
  stm.last = NULL;

  /* initialize ppv */

  ppv.next = NULL;

  /* load the simulation */

  errflag = 0;

  load_simulation(&strm,ee,nelm,nn,nnode,&qq,&nq,&ss,&ppv,&errflag);

  if (errflag != 0)
    {
      printf(" *** Load SIMULATION Section ABORTED ***\n");
      return 0;  /* don't load anything if there was an error */
    }

  /* free the old queue and print_var structures */

  for (temp_pv = pv->next; temp_pv != NULL ;)
    {
      old = temp_pv;
      temp_pv = temp_pv->next;
      free(old);
    }

  for (i = 0 ; i < *nbrq ; i++)
    free((*q)[i]);

  free(*q);
```

```
  /* assign the new structures */

  (*q) = qq;
  *nbrq = nq;

  pv->next = ppv.next;

  /* change the Simulate structure */

  simulate->dt             = ss.dt;
  simulate->tmin           = ss.tmin;
  simulate->tmax           = ss.tmax;
  simulate->time           = ss.time;
  simulate->max_iteration  = ss.max_iteration;
  simulate->converge       = ss.converge;
  simulate->delta          = ss.delta;
  simulate->delta_min      = ss.delta_min;
  simulate->print_dt       = ss.print_dt;

  return 0;


}

write_init(out,e,nelm,nn,nnode)
FILE *out;
ELEMENT **e;
int nelm;
NODE **nn;
int nnode;
{
  int i,j,k,ee,v;

  fprintf(out,"INITIALIZE\n!\n");

  for (i = 0 ; i < nelm ; i++)
    {
      /* print out present values of input variables */

      for (j = 0 ; j < e[i]->con.nbr_inputs ; j++)
        {
          fprintf(out,"  %20s : %20s %12.5g\n",e[i]->name,
                  e[i]->device->input_name[j],e[i]->con.in[j]);
        }

      /* print out present values of state variables */

      for (j = 0 ; j < e[i]->con.nbr_states ; j++)
        {
          fprintf(out,"  %20s : %20s %12.5g\n",e[i]->name,
                  e[i]->device->state_name[j],e[i]->con.state[j]);
        }
    }
```

```
  fprintf(out,"  END\n!\n");

  fprintf(out,"NODE VOLTAGE INITIALIZATION\n");

  for (i = 0 ; i < nnode ; i++)
    {
      /* print out present values of nodes */

      for (j = 0 ; j < nn[i]->nbr_subnode ; j++)
        {

          /* only write out those that are voltage subnodes */

          if (nn[i]->subnode[j]->type != 0 )
            continue;

          /* find pointers in element array */

          ee = nn[i]->subnode[j]->elm_ptr[0];
          v = nn[i]->subnode[j]->var_ptr[0];

          fprintf(out,"  %20s : %20s %12.5g\n", nn[i]->name ,
                  nn[i]->subnode[j]->name,e[ee]->con.in[v]);
        }
    }

  fprintf(out,"  END\n!\n");

  fprintf(out,"EXTERNAL INPUTS INITIALIZATION\n!\n");


  for (i = 0 ; i < nelm ; i++)
    {
      /* print out initial values of external inputs */

      for (j = 0 ; j < e[i]->con.nbr_ext_in ; j++)
        {
          fprintf(out,"  %20s : %20s %12.5g\n",e[i]->name,
                  e[i]->device->ext_in_name[j],e[i]->con.init_ext_in[j]);
        }
    }

  fprintf(out,"  END\n!\n");

  if (out != stdout) fclose(out);

  return 0;

}

load_init(in,filename,e,nelm,nn,nnode)
FILE *in;
ELEMENT **e;
```

```
int nelm;
NODE **nn;
int nnode;
char *filename;
{
  STREAM_PTR *strm,stm;
  int errflag,i,j;
  NODE **tn;
  ELEMENT **te;
  char *calloc();

  /* initialize the stream pointer */

  strm = &stm;

  stm.in = in;
  strcpy(stm.filename , filename);
  stm.line_nbr = 0;
  stm.last = NULL;

  /* allocate Node and Element Arrays */

  tn = (NODE **)   calloc((unsigned) nnode , sizeof(NODE *));
  te = (ELEMENT **)calloc((unsigned) nelm , sizeof(ELEMENT *));

  for (i = 0 ; i < nnode ; i++)
    tn[i] = (NODE *) calloc((unsigned) 1 , sizeof(NODE ));
  for (i = 0 ; i < nelm ; i++)
    te[i] = (ELEMENT *) calloc((unsigned) 1 , sizeof(ELEMENT));

  /* fill up the arrays */

  for ( i = 0 ; i < nelm ; i++)
    {
      te[i]->con.nbr_inputs = e[i]->con.nbr_inputs;
      te[i]->con.nbr_states = e[i]->con.nbr_states;
      te[i]->con.nbr_ext_in = e[i]->con.nbr_ext_in;

      te[i]->device = e[i]->device;
      te[i]->name   = e[i]->name;
      te[i]->flag   = e[i]->flag;
      te[i]->serial = e[i]->serial;

      if (te[i]->con.nbr_inputs)
        te[i]->con.init_in =
          (double *) calloc((unsigned)te[i]->con.nbr_inputs , sizeof(double));
      if (te[i]->con.nbr_states)
        te[i]->con.init_state =
          (double *) calloc((unsigned)te[i]->con.nbr_states , sizeof(double));
      if (te[i]->con.nbr_ext_in)
        te[i]->con.init_ext_in =
          (double *) calloc((unsigned)te[i]->con.nbr_ext_in , sizeof(double));
    }
```

```
  for (i = 0 ; i < nnode ; i++)
    {
      tn[i]->name = nn[i]->name;
      tn[i]->nbr_subnode = nn[i]->nbr_subnode;

      tn[i]->subnode = (SUBNODE **) calloc((unsigned) tn[i]->nbr_subnode,
                                           sizeof(SUBNODE *));
      for (j = 0 ; j < tn[i]->nbr_subnode ; j++)
        {
          tn[i]->subnode[j] = (SUBNODE *) calloc ((unsigned) 1,
                                                  sizeof (SUBNODE));

          tn[i]->subnode[j]->type = nn[i]->subnode[j]->type;
          tn[i]->subnode[j]->init_volt = 0.0;
          tn[i]->subnode[j]->name = nn[i]->subnode[j]->name;
        }
    }

  errflag = 0;

  i = load_initial(&strm,te,nelm,tn,nnode,"!",&errflag,1);

  /* see if the load had an error */

  if (i != 0 || errflag != 0)
    {
      printf(" *** Load INITIALIZATION Section  ABORTED ***\n");
      return 0;
    }

  /* save the results */

  for (i = 0 ; i < nelm ; i++)
    {
      if (e[i]->con.nbr_inputs > 0)
        {
          free(e[i]->con.init_in);
          e[i]->con.init_in = te[i]->con.init_in;
        }
      if (e[i]->con.nbr_states > 0)
        {
          free(e[i]->con.init_state);
          e[i]->con.init_state = te[i]->con.init_state;
        }
      if (e[i]->con.nbr_ext_in > 0)
        {
          free(e[i]->con.init_ext_in);
          e[i]->con.init_ext_in = te[i]->con.init_ext_in;
        }
      free(te[i]);
    }
  free(te);

  for (i = 0 ; i < nnode ; i++)
```

```
    {
      for (j = 0 ; j < nn[i]->nbr_subnode ; j++)
        {
          if (nn[i]->subnode[j]->ref_flag == 0) /* don't change ref subnode */
            nn[i]->subnode[j]->init_volt = tn[i]->subnode[j]->init_volt;
          free(tn[i]->subnode[j]);
        }
      free(te[i]);
    }
  free(te);

  return 0;


}



change_directory(filename)
char *filename;
{
  strstrip(filename);

  if (filename[0] == NULL)
    {
      printf(" Present Working Directory : ");
      fflush(stdout);

      /* The following line is system dependent */

      system("pwd");


      printf(" Enter New Directory : ");
      gets(filename);
      strstrip(filename);
      if (filename[0] == NULL) return;
    }

  if (chdir(filename) != 0)
    printf(" *** ERROR : change directory unsuccessful\n\n");
}
```

gauss_eliminate.c

```
/* gauss_eliminate.c */
/* Norbert H. Doerry

   14 February 1989

   This routine solves a general system of linear equations using
   Gaussian Elimination with partial pivoting.  The function returns
   a value of 0 if all went well, and a value of 1 if the matrix
   is singular

   14 Feb modifications:

   The a matrix is no longer preserved,  neither is the c matrix

*/
#include <stdio.h>
#define DEBUG 0


#define aa a
#define cc c

gauss_eliminate(n,a,c,x)
int n;     /* The dimension of the system */
double *a;  /* input matrix that is dimensioned n by n        */
double *c;  /* The right hand vector dimensioned n            */
double *x;  /* The solution to the system of linear equations */
{
  char *calloc();
  double pivot,temp;
  int i,j,k,pvt;

  /* print out the matrix if DEBUG is set */

  for (k = 0 ; k < n && DEBUG ; printf(" : %10.4f\n",c[k++]))
    for (j = 0; j < n ; j++)
      printf(" %3.0f",a[j*n + k]);

  if (DEBUG) printf("\n");

  /* the input arrays are not preserved */

  /* print out the matrix if DEBUG is set */

  for (k = 0 ; k < n && DEBUG ; printf(" : %10.4f\n",cc[k++]))
    for (j = 0; j < n ; j++)
      printf(" %3.0f",aa[j*n + k]);

  if (DEBUG) printf("\n");

  /* triangularize matrix */

  for (i = 0 ; i < n ; i++)
    {
```

```
    /* find pivot */

    pvt = i;
    pivot = aa[i + i*n]*aa[i + i*n];

    for (j = i+1 ; j < n ; j++)
      {
        if (DEBUG) printf("%7.4f  :: %7.4f\n",pivot,aa[j + i*n]*aa[j +i*n]);
        if (pivot < aa[j + i*n]*aa[j + i*n])
          {
            pivot = aa[j + i*n] * aa[j +i*n];
            pvt = j;
          }
      }

    pivot = aa[pvt + i*n];

    /* see if singular matrix */

    if (pivot == 0)
     return 1;

    /* switch rows if necessary */

    if (i != pvt)
      {
        for (j = i ; j < n ; j++)
          {
            temp          = aa[i   + j*n];
            aa[i   + j*n] = aa[pvt + j*n] / pivot; /* make pivot 1 */
            aa[pvt + j*n] = temp;
          }
        temp    = cc[i];
        cc[i]   = cc[pvt] / pivot;
        cc[pvt] = temp;
      }
    else  /* make pivot equal to one */
      {
        cc[i] /= pivot;
        for (j = i ; j < n ; j++)
          aa[i + j*n] /= pivot;
      }

    /* get zeros under pivot */

    for (j = i+1 ; j < n ; j++)
      {
        if ((temp = aa[j + i*n]) == 0) continue;
        for (k = i ; k < n ; k++)
          aa[j + k*n] -= temp * aa[i + k*n];
        cc[j] -= temp * cc[i];
      }

    /* print out the matrix if DEBUG is set */
```

```
      for (k = 0 ; k < n && DEBUG ; printf(" : %10.4f\n",cc[k++]))
        for (j = 0; j < n ; j++)
          printf(" %3.0f",aa[j*n + k]);

      if (DEBUG) printf("\n");
    }


  /* aa[] should be triangularized */

  /* print out the matrix if DEBUG is set */

  for (i = 0 ; i < n && DEBUG ; printf(" : %10.4f\n",cc[i++]))
    for (j = 0; j < n ; j++)
      printf(" %3.0f",aa[j*n + i]);
  if (DEBUG) printf("\n");

  /* back subsitute */

  for (i = n-1 ; i >= 0 ; i--)
    {
      x[i] = cc[i];

      for (j = i+1 ; j < n ; j++)
        x[i] -= aa[i + j*n] * x[j];
    }

  return 0;
}
```

M7-321-7.MIT.EDU:nhdoerry

stdin

Fri May 12 10:44:42 1989

ln03-7-321 / LN03R

```
ln03-7-321 M7-321-7.MIT.EDU:nhdoerry  Job: stdin  Date: Fri May 12 10:44:42 1989

ln03-7-321 M7-321-7.MIT.EDU:nhdoerry  Job: stdin  Date: Fri May 12 10:44:42 1989

ln03-7-321 M7-321-7.MIT.EDU:nhdoerry  Job: stdin  Date: Fri May 12 10:44:42 1989

ln03-7-321 M7-321-7.MIT.EDU:nhdoerry  Job: stdin  Date: Fri May 12 10:44:42 1989
```

```
/* integ.c */
/* Norbert H. Doerry

   18 October 1988


   This routine uses trapezoidal integration to integrate a variable
   of the form

   dx/dt = f(x(t),y(t),t)

   x(t) - x(t - dt) = [dt/2]*[f(x(t),y(t),t) + f(x(t-dt),y(t-dt),t-dt)]

   This routine returns G(x) which is :

   G(x) = x(t) - x(t-dt) - [dt/2]*[f(x(t),y(t),t) + f(x(t-dt),y(t-dt),t-dt)]

   This variable should be driven to zero with Newton Raphson in order
   to determine the proper new variable

*/

double integ(x,xold,fx,fxold,dt)
double x;             /* x(t)                              */
double xold;          /* x(t - dt)                         */
double fx;            /* f(x(t),y(t),t)                    */
double fxold;         /* f(x(t - dt), y(t - dt), t-dt)     */
double dt;            /* dt                                */
{
  return x - xold - (dt / 2.0) * (fx + fxold);
}
```

```
/* ioliba.c */
/* Norbert H. Doerry

   Last update : 25 March 1988

   This library contains a set of routines to augment the IO functions
   in the standard IO libraries.
*/
/* rev a:  10 July 88: fixed stofa                                    */
/* rev b:  11 July 88: added suctolc,slctouc,parse                    */
/* rev c:  24 Oct  88: added strsplit,strstrip,Stofa,revised getflta  */
/* rev d:  10 Nov  88: fixed getflta                                  */
/* rev e:  11 Nov  88: modified stofa, added stoda, Stoda             */
/* rev f:  25 Mar  89: added strextract                               */

#include <stdio.h>
#include <strings.h>
#define MAX_EXP 38          /* maximum sized exponent for system */


/* stofa */
/* converts a string to an array of floating point numbers            */
/* passes back the array and the number of numbers successfully       */
/* converted.  The returned value is a zero if read successfullly     */
/* to end of line, otherwise, returns the character that reading      */
/* failed at.  The third argument passed to the function is the       */
/* maximum number of elements in the array                            */
/* rev a:  10 July 88: fixed bug that caused extra number to be converted */
/* rev e:  11 Nov  88: Added exponential notation                     */

stofa(string,fltaptr,nbrptr)
char string[];
float fltaptr[];
int *nbrptr;

{
  int sign;
  int index = 0;
  float power ;
  int maxlen,i;
  char ch;
  float inch;
  int exponent,exp_sign;

  maxlen = *nbrptr;
  *nbrptr = 0;

  /* strip off leading blanks and tabs */

  while ((ch=string[index++]) == ' ' || ch == '\t');

  /* convert the numbers */

  while ((ch >= '0' && ch <= '9') || ch == '.' || ch == '+' || ch == '-'
```

```
        || ch == ',' || ch == ';' || ch == ':' || ch == 'e' || ch == 'E')
    {
      sign = 1;     /* default is positive */
      power = 10.0;
      exponent = 0;
      exp_sign = 1;

      if ( ch == '-' || ch == '+')
        {
          sign = (ch == '-') ? -1 : 1;
          ch = string[index++];
        }

      fltaptr[*nbrptr] = 0; /* initialize value */
      while (ch >= '0' && ch <= '9')
        {
          fltaptr[*nbrptr] *= 10.0;
          fltaptr[*nbrptr] += (float) (ch - '0');
          ch = string[index++];
        }
      if (ch == '.') /* check for decimal point */
        while ((ch = string[index++]) >= '0' && ch <= '9')
          {
            fltaptr[*nbrptr] += (float) (ch - '0') / power;
            power *= 10.0;
          }

      else if (ch == '-') /* check for ft-in entry */
        {
          if ((ch = string[index++]) >= '0' && ch <= '9')
            {
              inch = ch - '0';

              /* see if 11 or 12 inches */

              if (inch == 1 || inch == 0)
                if ((ch = string[index++]) == '0' || ch == '1')
                  {
                    inch *= 10.0;
                    inch += (float)(ch - '0');
                  }
                else
                  index--;

              power = 10.0;

              if ((ch = string[index++]) == '.') /* ft-decimal inch */
                {
                  while ((ch = string[index++]) >= '0' &&  ch <= '9')
                    {
                      inch += (float) (ch - '0') / power;
                      power *= 10;
                      }
                }
```

```
            else if (ch == '-') /* ft-inch-eighth */
              {
                if ((ch = string[index++]) >= '0' && ch <= '7')
                  {
                    inch += (float)(ch - '0') / 8.0;
                    if ((ch = string[index++]) == '.')
                      while ((ch = string[index++]) >= '0' &&
                             ch <= '9')
                        {
                          inch += (float) (ch - '0') / (8.0 * power);
                          power *= 10.0;
                        }
                  }
              }
            fltaptr[*nbrptr] += inch / 12.0;
          }
      }

    /* check for exponent */

    if (ch == 'e' || ch == 'E')
      {

        /* if the mantissa is not specified  but an exponent was specified
           set the mantissa to 1.0 */

        if (fltaptr[*nbrptr] == 0 && string[index - 2] != '0' &&
            string[index - 2] != '.')
          fltaptr[*nbrptr] = 1.0;

        /* get the next character */

        ch = string[index++];

        exp_sign = 1;
        power = 10;

        /* get the sign of the exponent */

        if ( ch == '-' || ch == '+')
          {
            exp_sign = (ch == '-') ? -1 : 1;
            ch = string[index++];
          }

        /* get the actual exponent value */

        exponent = 0; /* initialize value */
        while (ch >= '0' && ch <= '9')
            {
              exponent *= 10;
              exponent += ch - '0';
              ch = string[index++];
            }
```

```
    }

  /* ensure exponent is not too large */

  /* MAX_EXP is the max size of an exponent  (1e+MAX_EXP is legal)*/

  exponent *= exp_sign;

  while(fltaptr[*nbrptr] > 1.0)
    {
      fltaptr[*nbrptr] /= 10.0;
      exponent++;
    }

  while(fltaptr[*nbrptr] < 0.0)
    {
      fltaptr[*nbrptr] *= 10.0;
      exponent--;
    }

  exp_sign = (exponent < 0) ? -1 : 1;

  exponent = (exponent < 0) ? -exponent : exponent;

  if (exponent > MAX_EXP)
    {
      exponent = MAX_EXP;
      fltaptr[*nbrptr] = (exp_sign == -1) ? 0.0 : 1.0;
    }

  /* multiply number by its exponent */

  power = (exp_sign == -1) ? 0.1 : 10.0;

  for (i = 0 ; i < exponent ; i++)
    fltaptr[*nbrptr] *= power;

  /* multiply number by its sign */

  fltaptr[*nbrptr] *= sign;
  (*nbrptr)++;

  /* stop if converted maximum number of elements */

  if (*nbrptr == maxlen) break;

  /* see if illegal character following a legal number */

  if (ch == '-' || ch == '+' || ch == '8' || ch == '9' || ch == '.')
    break;

  /* ignore delimiting spaces */
```

```
      while (ch == ' ' || ch == '\t')
        ch = string[index++];

      /* ignore delimiting : ; , */

      if ( ch == ':' || ch == ';' || ch ==',' )
        while ((ch = string[index++]) == ' ' || ch == '\t');

      /* interpret successive : : , as zero entries in the array */

      while ( ch == ':' || ch == ';' || ch ==',' )
        {
          if (*nbrptr < maxlen)
            {
              fltaptr[*nbrptr] = 0;
              (*nbrptr)++;
            }
          else break;

          while((ch = string[index++]) == ' ' || ch == '\t');
        }

    }
  return (ch);
}

/* stoda */
/* converts a string to an array of double precision floating point numbers */
/* passes back the array and the number of numbers successfully             */
/* converted.  The returned value is a zero if read successfullly           */
/* to end of line, otherwise, returns the character that reading            */
/* failed at.  The third argument passed to the function is the             */
/* maximum number of elements in the array                                  */


stoda(string,fltaptr,nbrptr)
char string[];
double fltaptr[];
int *nbrptr;

{
  int sign;
  int index = 0;
  double power ;
  int maxlen,i;
  char ch;
  int exponent,exp_sign;

  maxlen = *nbrptr;
  *nbrptr = 0;

  /* strip off leading blanks and tabs */

  while ((ch=string[index++]) == ' ' || ch == '\t');
```

```
/* convert the numbers */

while ((ch >= '0' && ch <= '9') || ch == '.' || ch == '+' || ch == '-'
       || ch == ',' || ch == ';' || ch == ':' || ch == 'e' || ch == 'E')
  {
    sign = 1;     /* default is positive */
    power = 10.0;
    exponent = 0;
    exp_sign = 1;

    if ( ch == '-' || ch == '+')
      {
        sign = (ch == '-') ? -1 : 1;
        ch = string[index++];
      }

    fltaptr[*nbrptr] = 0; /* initialize value */
    while (ch >= '0' && ch <= '9')
      {
        fltaptr[*nbrptr] *= 10.0;
        fltaptr[*nbrptr] += (float) (ch - '0');
        ch = string[index++];
      }
    if (ch == '.') /* check for decimal point */
      while ((ch = string[index++]) >= '0' && ch <= '9')
        {
          fltaptr[*nbrptr] += (float) (ch - '0') / power;
          power *= 10.0;
        }

    /* check for exponent */

    if (ch == 'e' || ch == 'E')
      {

        /* if the mantissa is not specified  but an exponent was specified
           set the mantissa to 1.0 */

        if (fltaptr[*nbrptr] == 0 && string[index - 2] != '0' &&
            string[index - 2] != '.')
          fltaptr[*nbrptr] = 1.0;

        /* get the next character */

        ch = string[index++];

        exp_sign = 1;
        power = 10;

        /* get the sign of the exponent */

        if ( ch == '-' || ch == '+')
          {
```

```
            exp_sign = (ch == '-') ? -1 : 1;
            ch = string[index++];
          }

        /* get the actual exponent value */

        exponent = 0; /* initialize value */
        while (ch >= '0' && ch <= '9')
            {
              exponent *= 10;
              exponent += ch - '0';
              ch = string[index++];
            }
      }

    /* ensure exponent is not too large */
    /* MAX_EXP is the max size of an exponent  (1e+MAX_EXP is legal)*/

    exponent *= exp_sign;

    while(fltaptr[*nbrptr] > 1.0)
      {
        fltaptr[*nbrptr] /= 10.0;
        exponent++;
      }

    while(fltaptr[*nbrptr] < 0.0)
      {
        fltaptr[*nbrptr] *= 10.0;
        exponent--;
      }

    exp_sign = (exponent < 0) ? -1 : 1;

    exponent = (exponent < 0) ? - exponent : exponent;

    if (exponent > MAX_EXP)
      {
        exponent = MAX_EXP;
        fltaptr[*nbrptr] = (exp_sign == -1) ? 0.0 : 1.0;
      }

    /* multiply number by its exponent */

    power = (exp_sign == -1) ? 0.1 : 10.0;

    for (i = 0 ; i < exponent ; i++)
      fltaptr[*nbrptr] *= power;

    /* multiply number by its sign */

    fltaptr[*nbrptr] *= sign;
    (*nbrptr)++;
```

```
      /* stop if converted maximum number of elements */

      if (*nbrptr == maxlen) break;

      /* see if illegal character following a legal number */

      if (ch == '-' || ch == '+' || ch == '.')
        break;

      /* ignore delimiting spaces */

      while (ch == ' ' || ch == '\t')
        ch = string[index++];

      /* ignore delimiting : ; , */

      if ( ch == ':' || ch == ';' || ch ==',' )
        while ((ch = string[index++]) == ' ' || ch == '\t');

      /* interpret successive : : , as zero entries in the array */

      while ( ch == ':' || ch == ';' || ch ==',' )
        {
          if (*nbrptr < maxlen)
            {
              fltaptr[*nbrptr] = 0;
              (*nbrptr)++;
            }
          else break;

          while((ch = string[index++]) == ' ' || ch == '\t');
        }

    }
  return (ch);
}

/* Stofa */
/* NHD
   Stofa is like stofa except that maxnum is specified as a separate
   argument from nbrptr
*/
Stofa(inline,fltaptr,nbrptr,maxnum)
char *inline;
float *fltaptr;
int *nbrptr,maxnum;
{
  *nbrptr = maxnum;
  return stofa(inline,fltaptr,nbrptr);
}

/* Stoda */
/* NHD
   Stoda is like stoda except that maxnum is specified as a separate
```

```
   argument from nbrptr
*/
Stoda(inline,fltaptr,nbrptr,maxnum)
char *inline;
double *fltaptr;
int *nbrptr,maxnum;
{
  *nbrptr = maxnum;
  return stoda(inline,fltaptr,nbrptr);
}

/* getflta */
/* NHD
   Get a string from stdin, and convert to an array of floating
   point numbers.  Return a NULL if read to end of line, or
   return the character that the conversion failed in.

   rev c:  added maxnum as an argument
*/

#define LEN 131

getflta(fltaptr,nbrptr,maxnum)
float fltaptr[];          /* floating number array */
int *nbrptr;              /* pointer holding number of numbers converted */
int maxnum;               /* maximum size of fltaptr */

  {
    char inline[LEN];

    gets(inline);
    return(Stofa(inline,fltaptr,nbrptr,maxnum));
  }

/* fgetflta */
/* Norbert H Doerry
   31 March 1988

   Get a string from a file, and convert it to an array
   of floating point numbers.  Return a NULL if read to end of
   line,  or return with the character the conversion failed.
   Returns a -1 if an EOF received.

   rev c: added maxnum as an argument
*/

fgetflta(file,fltaptr,nbrptr,maxnum)
FILE *file;
float fltaptr[];
int *nbrptr,maxnum;

{
  char inline[LEN];
  int ans;
```

```
  if (fgets(inline,LEN,file) == NULL)
    return -1;

  ans = Stofa(inline,fltaptr,nbrptr,maxnum);
  if (ans == '\n') ans = NULL;
  return ans;

}

/* parse */
/* Norbert H Doerry
   11 July 1988

   Parse a string into its elements that are separated by spaces.
   maxlen is the maximum length of a parse element.
   mascnt is the maximum number of elements to parse.
   cnt is the number of elements parsed.
   array contains the parsed strings.

   rev c: also ignores tabs
*/

parse(string,array,maxlen,maxcnt,cnt)
char string[];
char array[];
int maxlen,maxcnt,*cnt;

{
  int i,j,k;

  i = j = k = 0;

  while (string[i] != NULL && k < maxcnt)
    {
      j = 0;

      /* strip off leading spaces and tabs */

      while (string[i] == ' ' || string[i] == '\t' ) i++;

      /* move characters to array */

      while (string[i] != ' ' && string[i] != NULL &&
             string[i] != '\t' && j < maxlen - 1)
        array[k*maxlen + j++] = string[i++];

      /* terminate array element with NULL */

      array[k*maxlen + j] = NULL;

      /* read to end of element if exceed maxlen */

      while (string[i] != ' ' && string[i] != '\t' && string[i] != NULL) i++;
```

```
      /* increment word counter */
      k++;
    }

  *cnt = k;

}


/* suctolc */
/* Norbert Doerry
   11 July 1988

   This converts all the upper case characters in a string to lower case

*/

suctolc(instring,outstring)
char instring[],outstring[];

{
  int i;

  for (i = 0; instring[i] != NULL ; i++)
    {
      if (instring[i] >= 'A' && instring[i] <= 'Z')
        outstring[i] = instring[i] - 'A' + 'a';
      else
        outstring[i] = instring[i];
    }
  outstring[i] = NULL;
}

/* slctouc */
/* NHD

   This converts all the lower case characters in a string to upper case

*/

slctouc(instring,outstring)
char instring[],outstring[];

{
  int i;

  for (i = 0; instring[i] != NULL ; i++)
    {
      if (instring[i] >= 'a' && instring[i] <= 'z')
        outstring[i] = instring[i] - 'a' + 'A';
      else
        outstring[i] = instring[i];
    }
```

```
  outstring[i] = NULL;
}


/* strcmpa */
/* Norbert H Doerry
   11 July 1988

   This is like strcmp, except that case of letters are not considered
*/
strcmpa(string1,string2)
char *string1,*string2;
{
  char *str1,*str2;
  char *malloc();
  int ans;

  str1 = malloc((unsigned) strlen(string1) + 1);
  str2 = malloc((unsigned) strlen(string2) + 1);

  slctouc(string1,str1);
  slctouc(string2,str2);
  ans = strcmp(str1,str2);
  free(str1);
  free(str2);
  return ans;
}

/* strncmpa */
/* Norbert H Doerry
   11 Juloy 1988

   This is like strncmp, except that case of letters are not considered
*/
strncmpa(string1,string2,n)
char *string1,*string2;
int n;

{
  char *str1,*str2;
  char *malloc();
  int ans;

  str1 = malloc((unsigned)strlen(string1) + 1);
  str2 = malloc((unsigned)strlen(string2) + 1);

  slctouc(string1,str1);
  slctouc(string2,str2);
  ans = strncmp(str1,str2,n);

  free(str1);
  free(str2);
  return ans;
}
```

```
/* strsplit */
/* This routine returns a substring of an input string.  The substring
   begins after n words are encountered in the input string.  It ends
   on the encounter of a new line character or the end of the input
   string.  len is the maximum length of string.
*/
strsplit(inline,s,n,len)
char *inline,*s;
int n,len;

{
  int i,j,k;

  i = 0;

  while (inline[i] == ' ' ) i++; /* strip off leading blanks */

  for (j = 0; j < n && inline[i] != NULL ; j++) /* read  in n words */
    {
      while (inline[i] != NULL && inline[i] != ' ' && inline[i] != '\t') i++;
      /* strip off trailing blanks and tabs */
      while (inline[i] == ' ' || inline[i] == '\t') i++;
    }
  k = 0;

  /* copy string */

  while (inline[i] != NULL && inline[i] != '\n' && k < len - 1)
    s[k++] = inline[i++];

  /* terminate with NULL */

  s[k] = NULL;

  /* strip off trailing blanks */

  for (j = strlen(s) - 1 ; s[j] == ' ' || s[j] == '\t' ||
       s[j] == '\n' ; s[j--] = NULL);

}

/* strstrip */

/* strstrip strips a string of leading and trailing spaces and tabs */

strstrip(s)
char *s;
{
  int i,j;

  /* find first none space or tab */
```

```
  for (i = 0 ; s[i] == ' ' || s[i] == '\t' ; i++);

  /* copy string */

  for (j = 0 ; s[i] != NULL ; s[j++] = s[i++]);

  s[j] = NULL;

  /* delete trailing spaces and tabs and Cr*/

  for (j = strlen(s) - 1 ; s[j] == ' ' || s[j] == '\t' ||
       s[j] == '\n' ; s[j--] = NULL);
}

/* strextract */
/* strextract returns the nth word in a string */

strextract(in,out,n,len)
char *in,*out; /* in is the input string , out is the output string    */
int n,len;     /* n is the desired word number , len is max len of out */
{
  int i,j;

  /* strip off the leading spaces and tabs */

  for (i = 0 ; in[i] == ' ' || in[i] == '\t' ; i++);

  /* ignore the first n-1 words */

  for (j = 0 ; j < n - 1 ; j++)
    {
      while (in[i] != ' ' && in[i] != '\t' && in[i] != NULL) i++;
      while (in[i] == ' ' || in[i] == '\t') i++;
    }

  /* copy the nth word */

  for (j = 0 ; j < len - 1 && in[i] != NULL && in[i] != ' ' && in[i] != '\t'
       && in[i] != '\n' ; j++ , i++)
    {
      out[j] = in[i];
    }

  out[j] = NULL;

}
```

load_device.c

```
/* load_device.c */
/* Norbert H. Doerry

   22 October 1988

   This routine loads in the device characteristics for a given number of
   devices.


*/
#include <stdio.h>
#include "doerry.h"

load_device(d,n,in,typ,d_function,d_name,ndev)
DEVICE **d;              /* array of pointers to device structures */
int n;                   /* number of device structures to read in */
int typ;                 /* beginning number used for type         */
FILE *in;                /* input file stream                      */
int (**d_function)();     /* array of pointers to device functions  */
char *d_name[];          /* array of device names                  */
int ndev;                /* total number of devices                */
{
  int i , j , k , ans , typa;

  ans = 0;
  typa = typ;

  /* read in data from file */

  for (i=0 ; i < n ; typa++ , i++)
    {
      j = read_device(d[i],in,typa);

      /* see if bad data */

      if (j == 1)
        {
          printf(" ERROR reading %d device : %s\n",typa,d[i]->name);
          ans = 1;
        }

      else if (j == 2)
        {
          printf(" EOF reached reading %d device : %s\n",typa,d[i]->name);
          return 2;
        }

      for (k = 0 ; strcmp(d[i]->name,d_name[k]) != 0 && k < ndev ; k++);

      if (k < ndev)    /* found the name */
        {
          d[i]->f = d_function[k];  /* copy the function address */
        }
      else
```

```
        {
          printf(" ERROR device %s is undefined : %s\n",d[i]->name);
          ans = 1;
        }

    }

  return ans;
}
```

load_element.c

```
/* load_element.c */
/* Norbert H. Doerry

   9 November 1988

   This file contains a function for loading the element descriptions
   from an input file.  The form of the element description is :


        DEVICE   ELEMENT
          PARAMETER_NAME  PARAMETER
               | |           | |
              \| |/         \| |/
               \ /           \ /
          END

   where

       DEVICE  is the name of the device type (i.e. resistor)
       ELEMENT is the specific element name (i.e. R1)
       PARAMETER NAME is the name of the DEVICE PARAMETER (i.e. R)
       PARAMETER is the value of the Parameter (i.e. 100 )

   additionally, an INCLUDE statement can be substituted for an
        Element Description.  An INCLUDE statement has the following
        form:

        INCLUDE filename

   where 'filename' is another file containing element descriptions

   *** Modified 9 January ***
       changed parameter for passing stream.  Now pass structure
       of STREAM_PTR instead of FILE.  This improves include file
       handling.

*/

/* The following structure is used to read in the elements because the
   total number of elements is not known until all the devices are read
   in.
*/

#include <stdio.h>
#include "doerry.h"

typedef struct Element_Ptr
{
  ELEMENT *e;                 /* present element */
  struct Element_Ptr *last;  /* pointer to last  structure holding element */
}
ELEMENT_PTR;
```

load_element.c

```
int load_element(strm1,ee,nelm,dev,ndev,errflag)
STREAM_PTR **strm1;
ELEMENT ***ee;      /* pointer to array of element descriptions */
int *nelm;        /* number of elements read in     */
DEVICE **dev;     /* array of device descriptions  */
int ndev;         /* number of devices in above array */
int *errflag;
{
  ELEMENT_PTR  *temp_elmnt , *elmnt;
  ELEMENT **e;
  int i,flag,serial;
  char *calloc();
  STREAM_PTR *temp_strm,*strm;
  char filename[MAXCHAR];

  /* initialize starting structures */

  elmnt = (ELEMENT_PTR *) calloc(1,sizeof(ELEMENT_PTR));
  elmnt->e =  (ELEMENT *) calloc(1,sizeof(ELEMENT));
  elmnt->last = NULL;          /* indicator that this is the first  stream */

  serial = 1;

  while (1)
    {
      strm = *strm1;

      flag = read_element(elmnt->e,strm,serial,dev,ndev);

      /* see if reached EOF */

      if (flag == 2)
        {
          if (strm->last == NULL)  /* read to the end of the first file */
            {
              free(strm);
              printf(" *** Error Line %d in file %s\n",
                     strm->line_nbr,strm->filename);
              printf("    *** EOF before NETWORK statement :\n\n");
              *errflag = 1;

              return 1;
            }

          /* read to the end of one of the include files */

          fclose(strm->in);

          /* bump back to last stream */

          temp_strm = strm;
          strm = strm->last;
          free(temp_strm);
```

```
        *strm1 = strm;

        continue;
      }

    /* see if unable to find device type */

    if (flag == -2)
      {

        /* see if include file */

        if (strncmpa(elmnt->e->name,"INCLUDE",3) == 0)
          {
            open_include(strm1,elmnt->e->name,errflag);
            continue;
          }

        /* see if end of section */

        if (strncmpa(elmnt->e->name,"NETWORK",7) == 0)
          {

            flag = 0;  /* successful load */

            break;
          }


        /* genuine bad device name */

        printf(" *** Error Line %d in file %s\n",
               strm->line_nbr,strm->filename);
        printf("    *** Unable to Interpret : %s ***\n",elmnt->e->name);
        *errflag = 1;
        continue;
      }

    /* read in an element */

    /* see if bad data encountered */

    if (flag == 1)
      {

        printf(" *** Error Line %d in file %s\n",
               strm->line_nbr,strm->filename);
        printf("    *** Encountered Bad Data Reading Element : %s ***\n",
               elmnt->e->name);
        *errflag = 1;
      }

    /* see if incomplete definition of parameters */
```

```
      if (flag == -1)
        {

          printf(" *** Error Line %d in file %s\n",
                 strm->line_nbr,strm->filename);
          printf("    *** Incomplete Parameters for Element : %s ***\n",
                 elmnt->e->name);
          *errflag = 1;
        }

      /* allocate the element structure block for the next element */

      temp_elmnt = elmnt;
      elmnt = (ELEMENT_PTR *) calloc((unsigned) 1,sizeof(ELEMENT_PTR));
      elmnt->last = temp_elmnt;
      elmnt->e    = (ELEMENT *) calloc((unsigned) 1,sizeof(ELEMENT));
      serial++;  /* increment serial number of element */

    }

  /* create element array */


  serial--;  /* serial is now the number of elements */

  *nelm = serial;

  e = (ELEMENT **)calloc((unsigned) serial,sizeof(ELEMENT *));
  *ee = e;

  /* create array of element pointers */

  for (i = 0 , elmnt = elmnt->last ; elmnt != NULL;i++)
    {
      e[i] = elmnt->e;
      temp_elmnt = elmnt;
      elmnt = elmnt->last;
      free(temp_elmnt);
    }

  *strm1 = strm;

  return 0;

}
```

load_initial.c

```
/* load_initial.c */
/* Norbert H. Doerry

   20 January 1989

   The following sections comprise the initialization section

   INITIALIZE
   EXTERNAL INPUTS INITIALIZATION
   NODE VOLTAGE INITIALIZATION

   The initialization section ends when the keyword SIMULATION is reached

   INITIALIZE
      each line of the initialization is of the form:

      ELEMENT : VARIABLE  VALUE

      where 'VARIABLE' can be a state or input  variable.
      the section ends with the keyword 'END'

   EXTERNAL INPUTS INITIALIZATION
      each line of the external input initialization is of the form:

      ELEMENT : EXTERNAL_INPUT  VALUE

      the section ends with the keyword 'END'

   NODE VOLTAGE INITIALIZATION
      each line of the node voltage initialization is of the form:

      NODE : SUBNODE  VALUE

      the section ends with the keyword 'END'

      NOTE :  this initialization is only necessary for Voltage Subnodes.
      If a  current subnode or a reference voltage subnode is specified,
      a warning is generated.


   The initialization section ends when the keyword SIMULATION is reached
   which indicates that the the next and final section starts.

*/
#include <stdio.h>
#include <math.h>
#include "doerry.h"

load_initial(strm1,e,nelm,nn,nnode,inline,errflag,eof)
STREAM_PTR **strm1;
ELEMENT **e;
int nelm;
char *inline;
int *errflag;
```

load_initial.c

```
NODE **nn;
int nnode;
int eof;  /* if eof = 0, then looks for 'SIMULATATION' to terminate section
             otherwise,  looks for the end of the file */
{
  int i,j,k,flag;
  STREAM_PTR *temp_strm,*strm;
  char *calloc();
  char filename[MAXCHAR];

  strstrip(inline);

  flag = 0;

  while(1)
    {

      if (strncmpa(inline,"INITIALIZE",7) == 0)
        flag = read_init(strm1,e,nelm,inline,errflag);

      else if (strncmpa(inline,"EXTERNAL",7) == 0)
        flag = read_ext_init(strm1,e,nelm,inline,errflag);

      else if (strncmpa(inline,"NODE VOLTAGE",7) == 0)
        flag = read_node_volt(strm1,nn,nnode,inline,errflag);

      else if (strncmpa(inline,"SIMULATION",7) == 0)
        {
          return 0;  /* read to end of initialization routine */
        }

      else if (strncmpa(inline,"INCLUDE",7) == 0)
        open_include(strm1,inline,errflag);

      else if (inline[0] != '!' && inline[0] != NULL && inline[0] != '#')
        {
          strm = *strm1;
          printf(" *** Error Line %d in file %s\n",
                 strm->line_nbr,strm->filename);
          printf("    *** INITIALIZE Syntax Error :\n %s\n\n",inline);
          *errflag = 1;
        }

      /* see if flag is one, which signifies that other routines ran out
         of file to read */

      if (flag != 0) return 1;

      /* read in next line */

      strm = *strm1;

      strm->line_nbr += 1;  /* increment line counter */
```

```
      while (fgets(inline,MAXCHAR,strm->in) == NULL)
        {
          /* Read to the end of the file, time to pop up one in the include
             stack */

          fclose(strm->in);

          if (strm->last == NULL) /* read back to the beginning */
            {
              free(strm);

              /* see if eof is set, if so, then done */

              if (eof) return 0;

              /* otherwise, reading to an end of file is an error */

              printf(" *** Error Line %d in file %s\n",
                     strm->line_nbr,strm->filename);
              printf("    *** EOF Reached before SIMULATION statement\n");
              *errflag = 1;
              return 1;
            }

          temp_strm = strm;
          strm = strm->last;
          free(temp_strm);

          *strm1 = strm;

        }
       strstrip(inline);

    }
}
open_include(strm1,inline,errflag)
STREAM_PTR **strm1;
char *inline;
int *errflag;
{
  char filename[MAXCHAR];
  STREAM_PTR *temp_strm,*strm;

  strm = *strm1;

  /* grab filename */

  strsplit(inline,filename,1,MAXCHAR);

  /* allocate new stream pointer structure */


  temp_strm = strm;  /* save present pointer */
  strm = (STREAM_PTR *) calloc((unsigned) 1,sizeof(STREAM_PTR));
```

```
  strm->last = temp_strm;
  strm->line_nbr = 0;
  strcpy(strm->filename,filename);

  /* try to open file */

  strm->in = fopen(filename,"r");

  /* see if unsuccessful */

  if (strm->in == NULL)
    {

      /* bump back to last stream */

      temp_strm = strm;
      strm = strm->last;
      free(temp_strm);

      printf(" *** Error Line %d in file %s\n",
             strm->line_nbr,strm->filename);

      printf("    *** Unable to Open Include File : %s\n",filename);
      *errflag = 1;
      return 1;
    }
  else
    {
      printf(" @@@ Successfully Opened Include File : %s\n",
             filename);
    }

  *strm1 = strm;
  return 0;

}

read_init(strm1,e,nelm,inline,errflag)
STREAM_PTR **strm1;
ELEMENT **e;
int nelm;
char *inline;
int *errflag;
{
  STREAM_PTR *strm,*temp_strm;
  char line[MAXCHAR];
  char e_name[MAXCHAR];
  char v_name[MAXCHAR];
  int i,j,k,ncnt;
  double value;

  while(1)
    {
```

```
      /* read in next line */

      strm = *strm1;

      strm->line_nbr += 1;  /* increment line counter */

      while (fgets(inline,MAXCHAR,strm->in) == NULL)
        {
          /* Read to the end of the file, time to pop up one in the include
             stack */

          fclose(strm->in);

          if (strm->last == NULL) /* read back to the beginning */
            {
              free(strm);
              printf(" *** Error Line %d in file %s\n",
                     strm->line_nbr,strm->filename);
              printf("    *** EOF Reached before END statement in %s\n",
                     "INITIALIZE");
              *errflag = 1;
              return 1;
            }

          temp_strm = strm;
          strm = strm->last;
          free(temp_strm);

          *strm1 = strm;

        }
      strstrip(inline);

      /* see if a comment line */

      if (inline[0] == NULL || inline[0] == '!' || inline[0] == '#')
        continue;

      /* see if end command */

      if (strncmpa(inline,"END",3) == 0)
        break;

      /* see if include file */

      if (strncmpa(inline,"INCLUDE",7) == 0)
        {
          open_include(strm1,inline,errflag);
          continue;
        }


      /* must be an element variable initialization */
```

```
      /* copy element name */

      for (i = 0 ; inline[i] != NULL && inline[i] != ':' ; i++)
        e_name[i] = inline[i];
      e_name[i] = NULL;
      strstrip(e_name);

      /* strip off tabs and spaces */

      for (i++; inline[i] == ' '  || inline[i] == '\t'; i++);

      /* copy variable name */

      for (j = 0 ; inline[i] != NULL && inline[i] != ' ' && inline[i] != '\t';
           i++,j++)
        v_name[j] = inline[i];
      v_name[j] = NULL;
      strstrip(v_name);

      /* copy value */

      for (j = 0 ; inline[i] != NULL ; i++,j++)
        line[j] = inline[i];
      line[j] = NULL;
      Stoda(line,&value,&ncnt,1);

      if (ncnt == 0) /* initialize to zero */
        value = 0.0;

      /* find the element */

      for (i = 0 ; i < nelm && strcmp(e_name,e[i]->name) != 0 ; i++);

      if (i >= nelm)  /* didn't find the element */
        {
          printf(" *** Error Line %d in file %s\n",
                 strm->line_nbr,strm->filename);
          printf("    *** ELEMENT Not Found Error (%s) :\n %s\n\n",
                 e_name,inline);
          *errflag = 1;
          continue;
        }

      /* find the state variable */

      for (j = 0 ; j < e[i]->device->nbr_states &&
           strcmp(v_name,e[i]->device->state_name[j]) != 0 ; j++);

      if (j < e[i]->device->nbr_states)
        {
          /* initialize the state variable */

          e[i]->con.init_state[j] = value;
          continue;
```

```
        }

      /* see if its an input variable */

      for (j = 0 ; j < e[i]->device->nbr_inputs &&
           strcmp(v_name,e[i]->device->input_name[j]) != 0 ; j++);

      if (j < e[i]->device->nbr_inputs)
        {
          /* initialize the input variable */

          e[i]->con.init_in[j] = value;
          continue;
        }

      /* can't recognize the variable */

      printf(" *** Error Line %d in file %s\n",
             strm->line_nbr,strm->filename);
      printf("    *** VARIABLE Not Found Error (%s) :\n %s\n\n",
             v_name,inline);
      *errflag = 1;

    }

  return 0;

}



read_ext_init(strm1,e,nelm,inline,errflag)
STREAM_PTR **strm1;
ELEMENT **e;
int nelm;
char *inline;
int *errflag;
{
  STREAM_PTR *strm,*temp_strm;
  char line[MAXCHAR];
  char e_name[MAXCHAR];
  char v_name[MAXCHAR];
  int i,j,k,ncnt;
  double value;

  while(1)
    {

      /* read in next line */

      strm = *strm1;

      strm->line_nbr += 1;  /* increment line counter */
```

```
    while (fgets(inline,MAXCHAR,strm->in) == NULL)
      {
        /* Read to the end of the file, time to pop up one in the include
           stack */

        fclose(strm->in);

        if (strm->last == NULL) /* read back to the beginning */
          {
            free(strm);
            printf(" *** Error Line %d in file %s\n",
                   strm->line_nbr,strm->filename);
            printf("    *** EOF Reached before END statement in %s\n",
                   "INITIALIZE");
            *errflag = 1;
            return 1;
          }

        temp_strm = strm;
        strm = strm->last;
        free(temp_strm);

        *strm1 = strm;

      }
    strstrip(inline);

    /* see if a comment */

    if (inline[0] == NULL || inline[0] == '!' || inline[0] == '#')
      continue;

    /* see if end command */

    if (strcmpa(inline,"END") == 0)
      break;

    /* see if include file */

    if (strcmpa(inline,"INCLUDE") == 0)
      {
        open_include(strm1,inline,errflag);
        continue;
      }


    /* must be an element variable initialization */

    /* copy element name */

    for (i = 0 ; inline[i] != NULL && inline[i] != ':' ; i++)
      e_name[i] = inline[i];
    e_name[i] = NULL;
    strstrip(e_name);
```

```
    /* strip off tabs and spaces */

    for (i++; inline[i] == ' '  || inline[i] == '\t'; i++);

    /* copy variable name */

    for (j = 0 ; inline[i] != NULL && inline[i] != ' ' && inline[i] != '\t';
         i++,j++)
      v_name[j] = inline[i];
    v_name[j] = NULL;
    strstrip(v_name);

    /* copy value */

    for (j = 0 ; inline[i] != NULL ; i++,j++)
      line[j] = inline[i];
    line[j] = NULL;
    Stoda(line,&value,&ncnt,1);

    if (ncnt == 0) /* initialize to zero */
      value = 0.0;

    /* find the element */

    for (i = 0 ; i < nelm && strcmp(e_name,e[i]->name) != 0 ; i++);

    if (i >= nelm)  /* didn't find the element */
      {
        printf(" *** Error Line %d in file %s\n",
               strm->line_nbr,strm->filename);
        printf("    *** ELEMENT Not Found Error (%s) :\n %s\n\n",
               e_name,inline);
        *errflag = 1;
        continue;
      }

    /* find the external input  variable */

    for (j = 0 ; j < e[i]->device->nbr_ext_in &&
         strcmp(v_name,e[i]->device->ext_in_name[j]) != 0 ; j++);

    if (j >= e[i]->device->nbr_ext_in)  /* didn't find the external input*/
      {
        printf(" *** Error Line %d in file %s\n",
               strm->line_nbr,strm->filename);
        printf("    *** EXTERNAL INPUT Not Found Error (%s) :\n %s\n\n",
               v_name,inline);
        *errflag = 1;
        continue;
      }

    /* initialize the state variable */
```

```
      e[i]->con.init_ext_in[j] = value;

    }

  return 0;

}

read_node_volt(strm1,nn,nnode,inline,errflag)
STREAM_PTR **strm1;
char *inline;
int *errflag;
NODE **nn;
int nnode;
{
  STREAM_PTR *strm,*temp_strm;
  char line[MAXCHAR];
  char n_name[MAXCHAR];
  char s_name[MAXCHAR];
  int i,j,k,ncnt;
  double value;

  while(1)
    {

      /* read in next line */

      strm = *strm1;

      strm->line_nbr += 1;  /* increment line counter */

      while (fgets(inline,MAXCHAR,strm->in) == NULL)
        {
          /* Read to the end of the file, time to pop up one in the include
             stack */

          fclose(strm->in);

          if (strm->last == NULL) /* read back to the beginning */
            {
              free(strm);
              printf(" *** Error Line %d in file %s\n",
                     strm->line_nbr,strm->filename);
              printf("    *** EOF Reached before END statement in %s\n",
                     "INITIALIZE");
              *errflag = 1;
              return 1;
            }

          temp_strm = strm;
          strm = strm->last;
          free(temp_strm);

          *strm1 = strm;
```

```
    }
  strstrip(inline);

  /* see if comment line */

  if (inline[0] == NULL || inline[0] == '!' || inline[0] == '#')
    continue;

  /* see if end command */

  if (strncmpa(inline,"END",3) == 0)
    break;

  /* see if include file */

  if (strncmpa(inline,"INCLUDE",7) == 0)
    {
      open_include(strm1,inline,errflag);
      continue;
    }


  /* must be an node voltage initialization */

  /* copy node name */

  for (i = 0 ; inline[i] != NULL && inline[i] != ':' ; i++)
    n_name[i] = inline[i];
  n_name[i] = NULL;
  strstrip(n_name);

  /* strip off tabs and spaces */

  for (i++; inline[i] == ' '  || inline[i] == '\t'; i++);

  /* copy subnode name */

  for (j = 0 ; inline[i] != NULL && inline[i] != ' ' && inline[i] != '\t';
       i++,j++)
    s_name[j] = inline[i];
  s_name[j] = NULL;
  strstrip(s_name);

  /* copy value */

  for (j = 0 ; inline[i] != NULL ; i++,j++)
    line[j] = inline[i];
  line[j] = NULL;
  Stoda(line,&value,&ncnt,1);

  if (ncnt == 0) /* initialize to zero */
    value = 0.0;
```

```
      /* find the node*/

      for (i = 0 ; i < nnode && strcmp(n_name,nn[i]->name) != 0 ; i++);

      if (i >= nnode)  /* didn't find the element */
        {
          printf(" *** Error Line %d in file %s\n",
                 strm->line_nbr,strm->filename);
          printf("    *** NODE Not Found Error (%s) :\n %s\n\n",
                 n_name,inline);
          *errflag = 1;
          continue;
        }

      /* find the subnode*/

      for (j = 0 ; j < nn[i]->nbr_subnode &&
           strcmp(s_name,nn[i]->subnode[j]->name) != 0 ; j++);

      if (j >= nn[i]->nbr_subnode)  /* didn't find the subnode */
        {
          printf(" *** Error Line %d in file %s\n",
                 strm->line_nbr,strm->filename);
          printf("    *** SUBNODE Not Found Error (%s) :\n %s\n\n",
                 s_name,inline);
          *errflag = 1;
          continue;
        }

      /* initialize the subnode */

      nn[i]->subnode[j]->init_volt = value;

      /* see if current subnode */

      if (nn[i]->subnode[j]->type == 1)
        {
          printf("*** WARNING Line %d in file %s\n",
                 strm->line_nbr,strm->filename);
          printf("   *** Initialization of Current Subnode Ignored :\n");
          printf(" %s\n\n",inline);
        }

      /* see if reference node */

      else if (nn[i]->subnode[j]->ref_flag == 1)
        {
          printf("*** WARNING Line %d in file %s\n",
                 strm->line_nbr,strm->filename);
          printf("   *** Initialization of Reference Voltage Subnode :\n");
          printf(" %s\n\n",inline);
        }
```

```
    }
  return 0;
}
```

```
/* load_network.c */
/* Norbert H. Doerry

   26 January 1989

  This file contains a function for loading the network descriptions
        from an input file.

        The network descriptions are read in until one of the following
        keywords is reached:

        INITIALIZE
        EXTERNAL (INPUTS INITIALIZATION)
        NODE VOLTAGE
        SIMULATION

        The INCLUDE keyword causes data to be taken from
        that file.

*/

#include <stdio.h>
#include "doerry.h"


int load_network(strm1,nn,nnode,e,nelm,errflag)
STREAM_PTR **strm1;
NODE ***nn;                    /* an array of pointers to NODES */
int *nnode;
ELEMENT **e;
int nelm;
int *errflag;
{
  STREAM_PTR *strm,*temp_strm;
  char *calloc();
  int i,j,k,l,m,mm,flag,serial,node_ctr;
  NODE *n,*new_node;
  char filename[MAXCHAR];

  strm = *strm1;

  n = (NODE *) calloc(1, sizeof(NODE));
  n->last = NULL;
  flag = 0;
  node_ctr = 0;

  while (1)
    {
      flag = read_network(strm,n,e,nelm,errflag);

      /* see if end of block */

      if (flag == -1 && (strncmpa(n->name,"INITIALIZE",7) == 0 ||
                         strncmpa(n->name,"EXTERNAL",7) == 0 ||
```

```
                    strncmpa(n->name,"SIMULATION",7) == 0 ||
                    strncmpa(n->name,"NODE VOLTAGE",7) == 0))
      {
        flag = 0;
        break;
      }

    /* see if include file */

    if (flag == -1 && strncmpa(n->name,"INCLUDE",7) == 0)
      {

        /* grab filename */

        strsplit(n->name,filename,1,MAXCHAR);

        /* allocate new stream pointer structure */


        temp_strm = strm;  /* save present pointer */
        strm = (STREAM_PTR *) calloc((unsigned) 1,sizeof(STREAM_PTR));
        strm->last = temp_strm;
        strm->line_nbr = 0;
        strcpy(strm->filename,filename);

        /* try to open file */

        strm->in = fopen(filename,"r");

        /* see if unsuccessful */

        if (strm->in == NULL)
          {

            /* bump back to last stream */

            temp_strm = strm;
            strm = strm->last;
            free(temp_strm);

            printf(" *** Error Line %d in file %s\n",
                   strm->line_nbr,strm->filename);

            printf("    *** Unable to Open Include File : %s\n",filename);
            *errflag = 1;
          }
        else
          {
            flag = 0;
            printf(" @@@ Successfully Opened Include File : %s\n",
                   filename);
          }

        continue;
```

```
    }

  if (flag == -1)
    {

      printf(" *** Error Line %d in file %s\n",
             strm->line_nbr,strm->filename);

      printf("    *** NETWORK Syntax Error :\n %s\n\n",n->name);
      free(n->name);
      *errflag = 1;
      continue;
    }

  /* See if read to end of file, if so, pop up one include file */

  if (flag == 2)
    {
      if (strm->last == NULL)  /* read to the end of the first file */
        {

          printf(" *** Error Line %d in file %s\n",
                 strm->line_nbr,strm->filename);

          printf("    *** EOF reached in NETWORK section:\n\n");
          break;
        }

      /* read to the end of one of the include files */

      fclose(strm->in);

      /* bump back to last stream */

      temp_strm = strm;
      strm = strm->last;
      free(temp_strm);

      flag = 0;
      continue;
    }


  /* allocate the new node */

  new_node = (NODE *) calloc(1,sizeof (NODE));

  if (new_node == NULL)
    {
      *errflag = 1;
      *strm1 = strm;

      printf(" *** Error Line %d in file %s\n",
             strm->line_nbr,strm->filename);
```

```
          printf("    *** Out of MEMORY\n");

          return 1;
        }

      new_node->last = n;
      n = new_node;

      /* increment node counter */
      node_ctr++;
    }

  if (flag)
    {
      *strm1 = strm;
      *errflag = 1;

      /* stop doing any more work if an error has been detected */

      if (flag == -3 || flag == 2)
        return 1;
      else
        return 0;
    }
  /* put the NODE structures into an array of pointers */

  *nnode = node_ctr;

  *nn = (NODE **) calloc(node_ctr + 1,sizeof(NODE *));

  (*nn)[node_ctr] = n;

  for (i = 0 ; i < node_ctr ; i++)
    {
      (*nn)[node_ctr - i - 1] = n->last;
      n = n->last;
    }

  /* check which elements are used and not used in network description */
  /* print out those elements which are not used */

  for (i = 0,j = 0 ; i < nelm ; i++) /* j is first time flag */
    {
      if (e[i]->flag == 1) continue; /* goto next element */

      if (j == 0)
        {
          printf("\n\n *** WARNING : The following Elements are defined");
          printf(" but not used : \n");
          j = 1;
        }

      printf("           %s\n",e[i]->name);
    }
```

```
  /* check to ensure that for all the elements that are used, all of their
     inputs are attached to a node. */


  for (i = 0; i < nelm ; i++) /* j is first time flag */
    {
      if (e[i]->flag == 0) continue; /* go to next element if not used */

      /* loop for each node variable */

      for (k = j = 0 ; k < e[i]->device->nbr_inputs ; k++,j = 0)
        {

          /* loop for each node */
          for (l = 0 ; l < *nnode ; l++)
            {
              /* loop for each subnode */

              for (m = 0 ; m < (*nn)[l]->nbr_subnode ; m++)
                {
                  /* loop for each connection in each subnode */

                  for (mm = 0 ; mm < (*nn)[l]->subnode[m]->nbr_connect ; mm++)
                    {

                      if (strcmp(e[i]->name,
                                 (*nn)[l]->subnode[m]->element[mm]) != 0)
                        continue; /* go on if element names don't match */

                      if (strcmp(e[i]->device->input_name[k],
                                 (*nn)[l]->subnode[m]->variable[mm]) != 0)
                        continue; /* go on if variable names don't match */

                      j++;  /* j is the number of nodes an input variable
                               is attached to */
                    }
                }
            }

          if (j == 0)
            {
              printf(" *** ERROR : Input Variable not attached to a node :\n");
              printf("    *** Element : %s || Input Variable : %s ***\n",
                     e[i]->name,
                     e[i]->device->input_name[k]);
              *errflag = 1;
            }
          else if (j != 1)
            {
              printf(" *** ERROR : Input Variable attached to %d nodes :\n",
                     j);
              printf("    *** Element : %s || Input Variable : %s ***\n",
                     e[i]->name,
```

```
                    e[i]->device->input_name[k]);
              *errflag = 1;
            }

        }

    }

 *strm1 = strm;
 return flag;
}
```

```
/* load_simulation.c */
/* Norbert H. Doerry

   27 January 1989

   This routine loads in all the information required to run the simulation
   of the program.  The Display section lists all the external output
   variables that may be printed out.

   TIME_STEP command sets the time step increment.
   TMIN command sets the starting time of the simulation.
   TMAX command sets the ending time of the simulation.
   DELTA is the fractional part of a variable that is used in calculating
               the  jacobian.
   DELTA_MIN is the minimum change in a variable for calculating the
               jacobian in case the variable is very small.
   PRINT_STEP sets the time increment for printing the values of the
               external variables.

   MAX_ITERATION is the maximum number of iterations of the Newton-Raphson
               method used before a failure to converge error is generated

   CONVERGE is the maximum mean square error of the implicit vector that
               is allowed for a balanced solution.

   The  Reference Section sets the voltage and current nodes and subnodes
   that are to be used as references.  The reference voltage node is
   always set equal to specified voltage (default is zero volts).
   The reference current node is not used to create a current law equation
   (This prevents a singular matrix)

   The External Input Command specifies the values of different
   external inputs at different times.

   The format is:

   SIMULATION

   DISPLAY
      ELEMENT : EXTERNAL_OUTPUT_VARIABLE
        ||                ||
        ||                ||
       \||/              \||/
        \/                \/
      END

   TIME_STEP       VALUE

   TMIN            VALUE
   TMAX            VALUE

   PRINT_STEP      VALUE

   DELTA           VALUE
```

load_simulation.c

```
   DELTA_MIN       VALUE

   REFERENCE
     V : NODE : SUBNODE VALUE
     I : NODE : SUBNODE
     END

   MAX_ITERATION   VALUE
   CONVERGE        VALUE


   EXTERNAL INPUTS
     ELEMENT : EXTERNAL_INPUT_VARIABLE  VALUE   TIME
        ||                ||             ||      ||
        ||                ||             ||      ||
       \||/              \||/           \||/    \||/
        \/                \/             \/      \/
      END

*/

#include <stdio.h>
#include <math.h>
#include "doerry.h"

#define DEBUG 0

load_simulation(strm1,e,nelm,nn,nnode,q,nbrq,simulate,pv,errflag)
STREAM_PTR **strm1;   /* pointer to pointer of current stream structure */
ELEMENT **e;  /* array of pointers to element structures                */
int nelm;     /* number of elements in element array                    */
NODE **nn;    /* array of pointers to node structures                   */
int nnode;    /* number of elements in node array                       */
QUEUE ***q;   /* pointer to an array of pointers to queue structures    */
int *nbrq;    /* number of elements in array of pointers to queue structures */
SIMULATE *simulate;  /* structure for simulation commands */
PRINT_VAR *pv;/* pointer to first PRINT_VAR structure  */
int *errflag; /* error flag, if = 1, cannot run simulation              */
{
  int i,j,jj,k,flag;
  STREAM_PTR *temp_strm,*strm;
  char *calloc();
  char inline[MAXCHAR];
  QUEUE **qq,*q_temp;
  double temp;

  qq = (QUEUE **) calloc(1, sizeof(QUEUE *));
  *qq = (QUEUE *) calloc(1, sizeof(QUEUE  ));

  (*qq)->last = NULL;  /* signal that this is first queue */

  jj = 1;
  flag = 0;  /* flag for ran out of file too soon */
  while(jj)
```

```
    {
      /* read in next line */

      strm = *strm1;

      strm->line_nbr += 1;  /* increment line counter */

      while (fgets(inline,MAXCHAR,strm->in) == NULL)
        {
          /* Read to the end of the file, time to pop up one in the include
             stack */

          fclose(strm->in);

          if (strm->last == NULL) /* read back to the beginning, we are done */
            {
              free(strm);
              jj = 0;
              break;
            }

          temp_strm = strm;
          strm = strm->last;
          free(temp_strm);

          *strm1 = strm;

        }

      if (jj == 0) break;  /* exit loop if done */

      strstrip(inline);

      /* see if line is a comment */

      if (inline[0] == '!' || inline[0] == '#' || inline[0] == NULL)
        continue;

      /* see if a valid command */

      if (strncmpa(inline,"DISPLAY",7) == 0)
        flag = read_display(strm1,e,nelm,nn,nnode,pv,errflag);

      else if (strncmpa(inline,"TIME_STEP",7) == 0)
        read_value(strm1,&(simulate->dt),inline,errflag);

      else if (strncmpa(inline,"TMIN",4) == 0)
        read_value(strm1,&(simulate->tmin),inline,errflag);

      else if (strncmpa(inline,"TMAX",4) == 0)
        read_value(strm1,&(simulate->tmax),inline,errflag);

      else if (strncmpa(inline,"PRINT_STEP",7) == 0)
        read_value(strm1,&(simulate->print_dt),inline,errflag);
```

```
      else if (strncmpa(inline,"DELTA_MIN",8) == 0)
        read_value(strm1,&(simulate->delta_min),inline,errflag);

      else if (strncmpa(inline,"DELTA",4) == 0)
        read_value(strm1,&(simulate->delta),inline,errflag);

      else if (strncmpa(inline,"CONVERGE",7) == 0)
        read_value(strm1,&(simulate->converge),inline,errflag);

      else if (strncmpa(inline,"MAX_ITERATION",7) == 0)
        {
          read_value(strm1,&temp,inline,errflag);
          simulate->max_iteration = (int) temp;
        }

      else if (strncmpa(inline,"REFERENCE",7) == 0)
        flag = read_reference(strm1,simulate,nn,nnode,errflag);

      else if (strncmpa(inline,"EXTERNAL INPUTS",7) == 0)
        flag = read_external(strm1,qq,e,nelm,errflag);

      else if (strncmpa(inline,"INCLUDE",7) == 0)
        open_include(strm1,inline,errflag);

      if (flag) return 1;  /* ran out of file too soon */

    }

  /* find out how many queue structures we have */

  q_temp = *qq;

  for (i = 0 ; q_temp->last != NULL ; q_temp = q_temp->last , i++);

  *nbrq = i;

  if (i == 0) return 0;  /* return if no queue structures */

  /* allocate array  for the queue structures */

  *q = (QUEUE **)calloc(i,sizeof(QUEUE *));

  /* see if out of memory */

  if (*q == NULL)
    {
      printf(" *** OUT of MEMORY ERROR ***\n");
      *errflag = 1;
      return 1;
    }

  /* store pointers in array */
```

```
  for (q_temp = *qq , i = 0 ;
       q_temp->last != NULL ;
       q_temp = q_temp->last , i++)
    (*q)[i] = q_temp->last;

  /* sort array by time (This is a bubble sort) */

  flag = 1;
  while (flag == 1)
    {
      flag = 0;
      for (i = 1 ; i < *nbrq ; i++)
        {
          if ( (*q)[i]->time >= (*q)[i-1]->time )
            continue;   /* in proper order */

          /* must switch two entries around */

          q_temp = (*q)[i];
          (*q)[i] = (*q)[i-1];
          (*q)[i-1] = q_temp;

          /* set flag to 1 to continue checking */

          flag = 1;
        }
    }
/*
  if (DEBUG)
    {
      for (i = 0 ; i < *nbrq ; i++)
        printf("elm = %d , var = %d , val = %f , time = %f\n",(*q)[i]->elm,
               (*q)[i]->var, (*q)[i]->value , (*q)[i]->time);
    }

*/
  return 0;

}


read_value(strm1,val,inline,errflag)
STREAM_PTR **strm1;
double *val;
char *inline;
int *errflag;
{
  STREAM_PTR *strm;
  char line[MAXCHAR];
  int ncnt,i;
  double value;

  strm = *strm1;
```

```
  /* strip off the command */

  strsplit(inline,line,1,MAXCHAR);

  /* strip line */

  strstrip(line);

  /* convert the number */

  Stoda(line,&value,&ncnt,1);

  /* set the initialization value */

  if (value >= 0 && ncnt == 1)
    *val = value;

  else
    {
      printf(" *** Error Line %d in file %s\n",
             strm->line_nbr,strm->filename);
      printf("    *** SYNTAX ERROR:\n %s\n\n",inline);
      *errflag = 1;
    }


  /* print results if debug */

  if (DEBUG)
    printf("inline = %s : line = %s : value = %f\n",inline,line,value);



}


read_display(strm1,e,nelm,nn,nnode,pv,errflag)
STREAM_PTR **strm1;   /* pointer to pointer of current stream structure */
ELEMENT **e;  /* array of pointers to element structures                */
int nelm;     /* number of elements in element array                    */
NODE **nn;    /* node array                                             */
int nnode;    /* number of elements in node array                       */
PRINT_VAR *pv;/* pointer to first PRINT_VAR structure in chain          */
int *errflag; /* error flag, if = 1, cannot run simulation              */
{
  int i,j,k,flag;
  STREAM_PTR *temp_strm,*strm;
  char *calloc();
  char v_name[MAXCHAR],e_name[MAXCHAR];
  char inline[MAXCHAR];
  PRINT_VAR *temp;

  while(1)
    {
```

```
      /* read in next line */

      strm = *strm1;

      strm->line_nbr += 1;  /* increment line counter */

      while (fgets(inline,MAXCHAR,strm->in) == NULL)
        {
          /* Read to the end of the file, time to pop up one in the include
             stack */

          fclose(strm->in);

          if (strm->last == NULL) /* read back to the beginning */
            {
              free(strm);
              printf(" *** Error Line %d in file %s\n",
                     strm->line_nbr,strm->filename);
              printf("    *** EOF reached in DISPLAY :\n %s\n\n",inline);
              *errflag = 1;


              return 1;
            }

          temp_strm = strm;
          strm = strm->last;
          free(temp_strm);

          *strm1 = strm;

        }

      strstrip(inline);

      /* see if line is a comment */

      if (inline[0] == '!' || inline[0] == '#' || inline[0] == NULL)
        continue;

      /* see if done */

      if (strncmpa(inline,"END",3) == 0)
        return 0;

      /* see if include file */

      else if (strncmpa(inline,"INCLUDE",7) == 0)
        open_include(strm1,inline,errflag);

      /* must be an element variable descripton */

      /* copy element name */
```

```
      for (i = 0 ; inline[i] != NULL && inline[i] != ':' ; i++)
        e_name[i] = inline[i];
      e_name[i] = NULL;
      strstrip(e_name);

     /* strip off tabs and spaces */

      for (i++; inline[i] == ' '  || inline[i] == '\t'; i++);

      /* copy variable name */

      for (j = 0 ; inline[i] != NULL && inline[i] != ' ' && inline[i] != '\t';
         i++,j++)
        v_name[j] = inline[i];
      v_name[j] = NULL;
      strstrip(v_name);

      /* find the element */

      for (i = 0 ; i < nelm && strcmp(e_name,e[i]->name) != 0 ; i++);

      /* found the element */

      if (i < nelm)
        {

          /* find the external output variable */

          for (j = 0 ; j < e[i]->device->nbr_ext_out &&
               strcmp(v_name,e[i]->device->ext_out_name[j]) != 0 ; j++);

          if (j < e[i]->device->nbr_ext_out)  /* found external output */
            {
              /* allocate and insert structure */

              for (temp = pv; temp->next != NULL ; temp = temp->next);

              temp->next  =
                (PRINT_VAR *) calloc((unsigned) 1 , sizeof(PRINT_VAR));
              temp  = temp->next;
              temp->next = NULL;

              /* store the information */

              temp->e = i;
              temp->v = j;
              temp->typ = 0;

              continue;
            }

          /* look for external input */
```

```
        for (j = 0 ; j < e[i]->device->nbr_ext_in &&
             strcmp(v_name,e[i]->device->ext_in_name[j]) != 0 ; j++);

        if (j < e[i]->device->nbr_ext_in)  /* found external output */
          {
            /* allocate and insert structure */

            for (temp = pv; temp->next != NULL ; temp = temp->next);

            temp->next  =
              (PRINT_VAR *) calloc((unsigned) 1 , sizeof(PRINT_VAR));
            temp  = temp->next;
            temp->next = NULL;

            /* store the information */

            temp->e = i;
            temp->v = j;
            temp->typ = 1;

            continue;
          }


        /* couldn't find external input or output */

        printf(" *** Error Line %d in file %s\n",
               strm->line_nbr,strm->filename);
        printf("    *** EXTERNAL VARIABLE Not Found Error (%s) :\n",
               v_name);
        printf(" %s\n\n",inline);
        *errflag = 1;
        continue;

      }


    /* find the node */

    for (i = 0 ; i < nnode && strcmp(e_name,nn[i]->name) != 0 ; i++);

    /* found the node */

    if (i < nnode)
      {
        /* look for the subnode */

        for (j = 0 ; j < nn[i]->nbr_subnode &&
             strcmp(v_name , nn[i]->subnode[j]->name) != 0 ; j++);

        if (j < nn[i]->nbr_subnode) /* found the subnode */
          {

            /* make sure subnode is a voltage subnode */
```

```
            if (nn[i]->subnode[j]->type == 1)
              {
                printf(" *** Error Line %d in file %s\n",
                       strm->line_nbr,strm->filename);
                printf("    *** SUBNODE Not of Voltage Type (%s) :\n",
                       v_name);
                printf(" %s\n\n",inline);
                *errflag = 1;
                continue;
              }

            /* allocate and insert structure */

            for (temp = pv; temp->next != NULL ; temp = temp->next);

            temp->next  =
              (PRINT_VAR *) calloc((unsigned) 1 , sizeof(PRINT_VAR));
            temp  = temp->next;
            temp->next = NULL;

            /* store the information */

            temp->e = i;
            temp->v = j;
            temp->typ = 2;

            continue;
          }

        /* couldn't find subnode*/

        printf(" *** Error Line %d in file %s\n",
               strm->line_nbr,strm->filename);
        printf("    *** SUBNODE Not Found Error (%s) :\n",
               v_name);
        printf(" %s\n\n",inline);
        *errflag = 1;
        continue;
      }

    /* didn't find the element */

    printf(" *** Error Line %d in file %s\n",
           strm->line_nbr,strm->filename);
    printf("    *** ELEMENT / NODE Not Found Error (%s) :\n %s\n\n",
           e_name,inline);
    *errflag = 1;
    continue;

  }

}
```

```
read_external(strm1,qq,e,nelm,errflag)
STREAM_PTR **strm1;   /* pointer to pointer of current stream structure */
QUEUE **qq;   /* pointer to pointer of current queue structure          */
ELEMENT **e;  /* array of pointers to element structures                */
int nelm;     /* number of elements in element array                    */
int *errflag; /* error flag, if = 1, cannot run simulation              */
{
  int i,j,k,flag,ncnt;
  STREAM_PTR *temp_strm,*strm;
  char *calloc();
  char inline[MAXCHAR];
  char line[MAXCHAR];
  char e_name[MAXCHAR];
  char v_name[MAXCHAR];
  double value[2];
  QUEUE *q_temp;

  flag = 0;

  while(1)
    {

      /* read in next line */

      strm = *strm1;

      strm->line_nbr += 1;  /* increment line counter */

      while (fgets(inline,MAXCHAR,strm->in) == NULL)
        {
          /* Read to the end of the file, time to pop up one in the include
             stack */

          fclose(strm->in);

          if (strm->last == NULL) /* read back to the beginning. */
            {
              free(strm);
              printf(" *** Error Line %d in file %s\n",
                     strm->line_nbr,strm->filename);
              printf("    *** EOF reached in EXTERNAL INPUT :\n %s\n\n",
                     inline);
              *errflag = 1;

              return 1;
            }

          temp_strm = strm;
          strm = strm->last;
          free(temp_strm);

          *strm1 = strm;
```

```
    }

  strstrip(inline);

  /* see if line is a comment */

  if (inline[0] == '!' || inline[0] == '#' || inline[0] == NULL)
    continue;

  /* see if done */

  if (strncmpa(inline,"END",3) == 0)
    break;

  /* see if include file */

  else if (strncmpa(inline,"INCLUDE",7) == 0)
    open_include(strm1,inline,errflag);


  /* must be an element variable descripton */

  /* copy element name */

  for (i = 0 ; inline[i] != NULL && inline[i] != ':' ; i++)
    e_name[i] = inline[i];
  e_name[i] = NULL;
  strstrip(e_name);

 /* strip off tabs and spaces */

  for (i++; inline[i] == ' '  || inline[i] == '\t'; i++);

  /* copy variable name */

  for (j = 0 ; inline[i] != NULL && inline[i] != ' ' && inline[i] != '\t';
     i++,j++)
    v_name[j] = inline[i];
  v_name[j] = NULL;
  strstrip(v_name);

  /* copy values */

  for (j = 0 ; inline[i] != NULL ; i++,j++)
    line[j] = inline[i];
  line[j] = NULL;
  Stoda(line,value,&ncnt,2);

  if (ncnt == 0) /* initialize to zero */
    value[0] = value[1] = 0.0;
  else if (ncnt == 1)
    value[1] = 0.0;
```

```
      /* find the element */

      for (i = 0 ; i < nelm && strcmp(e_name,e[i]->name) != 0 ; i++);

      if (i >= nelm)  /* didn't find the element */
        {
         printf(" *** Error Line %d in file %s\n",
              strm->line_nbr,strm->filename);
         printf("    *** ELEMENT Not Found Error (%s) :\n %s\n\n",
                e_name,inline);
         *errflag = 1;
         continue;
        }

      /* find the external input variable */

      for (j = 0 ; j < e[i]->device->nbr_ext_in &&
         strcmp(v_name,e[i]->device->ext_in_name[j]) != 0 ; j++)

      if (j >= e[i]->device->nbr_ext_in)  /* didn't find the external input */
        {
         printf(" *** Error Line %d in file %s\n",
                strm->line_nbr,strm->filename);
         printf("    *** EXTERNAL INPUT VARIABLE Not Found Error (%s) :\n",
                v_name);
         printf(" %s\n\n",inline);
         *errflag = 1;
         continue;
        }

      /* store in queue */

      (*qq)->elm = i;
      (*qq)->var = j;
      (*qq)->value = value[0];
      (*qq)->time = value[1];

      /* allocate new queue structure */

      q_temp = (QUEUE *) calloc(1,sizeof(QUEUE));
      q_temp->last = *qq;
      *qq = q_temp;

    }



}



read_reference(strm1,simulate,nn,nnode,errflag)
STREAM_PTR **strm1;
```

load_simulation.c

```
SIMULATE *simulate;
NODE **nn;
int nnode;
int *errflag;
{
  STREAM_PTR *strm,*temp_strm;
  char inline[MAXCHAR],line[MAXCHAR],n_name[MAXCHAR],s_name[MAXCHAR];
  int flag,i,j,ncnt;
  double val;

  while(1)
    {

      /* read in next line */

      strm = *strm1;

      strm->line_nbr += 1;  /* increment line counter */

      while (fgets(inline,MAXCHAR,strm->in) == NULL)
      {
        /* Read to the end of the file, time to pop up one in the include
           stack */

        fclose(strm->in);

        if (strm->last == NULL) /* read back to the beginning */
          {
            free(strm);
            printf(" *** Error Line %d in file %s\n",
                   strm->line_nbr,strm->filename);
            printf("    *** EOF reached in REFERENCE :\n %s\n\n",inline);
            *errflag = 1;


            return 1;
          }

        temp_strm = strm;
        strm = strm->last;
        free(temp_strm);

        *strm1 = strm;

      }

      strstrip(inline);

      /* see if line is a comment */

      if (inline[0] == '!' || inline[0] == '#' || inline[0] == NULL)
      continue;

      /* see if done */
```

```
    if (strncmpa(inline,"END",3) == 0)
    return 0;

    /* see if include file */

    else if (strncmpa(inline,"INCLUDE",7) == 0)
    open_include(strm1,inline,errflag);

    /* must be the reference voltage or current node */

    /* see if voltage node */

    strcpy(line,inline);

    if (inline[0] == 'v' || inline[0] == 'V')
      {
        /* strip off v and colon */

        for (i = 0 ; line[i] != ':' && line[i] != NULL ; i++)
          line[i] = ' ';
        line[i] = ' ';

        strstrip(line);

        /* grap the node name */

        for (i = 0 ; line[i] != ':' && line[i] != NULL; i++)
          n_name[i] = line[i];
        n_name[i] = NULL;

        strstrip(n_name);

        /* grap the subnode name */

        if (line[i] != NULL) i++;
        while (line[i] == ' ' || line[i] == '\t') i++;
        for (j = 0  ; line[i] != NULL && line[i] != ' ' && line[i] != '\t'
             ; i++,j++)
          s_name[j] = line[i];
        s_name[j] = NULL;

        strstrip(s_name);

        /* find the value */

        Stoda(line+i , &val , &ncnt, 1);

        if (ncnt == 0) val = 0;


        /* find the node and subnode */

        for (i = 0 ; i < nnode ; i++)
```

```
        if (strcmp(n_name,nn[i]->name) == 0) break;

      if (i == nnode)  /* didn't find the node */
        {
          printf(" *** Error Line %d in file %s\n",
                 strm->line_nbr,strm->filename);
          printf("    *** NODE not found ERROR:\n %s\n\n",inline);
          *errflag = 1;
          continue;
        }

      for (j = 0 ; j < nn[i]->nbr_subnode ; j++)
        if (strcmp(s_name,nn[i]->subnode[j]->name) == 0) break;

      if (j == nn[i]->nbr_subnode) /* didn't find the subnode */
        {
          printf(" *** Error Line %d in file %s\n",
                 strm->line_nbr,strm->filename);
          printf("    *** SUBNODE not found ERROR:\n %s\n\n",inline);
          *errflag = 1;
          continue;
        }

      if (nn[i]->subnode[j]->type != 0) /* Not a voltage  subnode */
        {
          printf(" *** Error Line %d in file %s\n",
                 strm->line_nbr,strm->filename);
          printf("    *** SUBNODE of wrong type:\n %s\n\n",inline);
          *errflag = 1;
          continue;
        }

      nn[i]->subnode[j]->ref_flag = 1;
      nn[i]->subnode[j]->init_volt = val;
    }

  /* see if current node */

  else if (inline[0] == 'i' || inline[0] == 'I')
    {
      /* strip off i and colon */

      for (i = 0 ; line[i] != ':' && line[i] != NULL ; i++)
        line[i] = ' ';
      line[i] = ' ';

      strstrip(line);

      /* grap the node name */

      for (i = 0 ; line[i] != ':' && line[i] != NULL; i++)
        n_name[i] = line[i];
      n_name[i] = NULL;
```

```
        strstrip(n_name);

        /* grap the subnode name */

        if (line[i] != NULL) i++;
        for (j = 0  ; line[i] != NULL ; i++,j++)
          s_name[j] = line[i];
        s_name[j] = NULL;

        strstrip(s_name);

        /* find the node and subnode */

        for (i = 0 ; i < nnode ; i++)
          if (strcmp(n_name,nn[i]->name) == 0) break;

        if (i == nnode)  /* didn't find the node */
          {
            printf(" *** Error Line %d in file %s\n",
                   strm->line_nbr,strm->filename);
            printf("    *** NODE not found ERROR:\n %s\n\n",inline);
            *errflag = 1;
            continue;
          }

        for (j = 0 ; j < nn[i]->nbr_subnode ; j++)
          if (strcmp(s_name,nn[i]->subnode[j]->name) == 0) break;

        if (j == nn[i]->nbr_subnode) /* didn't find the subnode */
          {
            printf(" *** Error Line %d in file %s\n",
                   strm->line_nbr,strm->filename);
            printf("    *** SUBNODE not found ERROR:\n %s\n\n",inline);
            *errflag = 1;
            continue;
          }

        if (nn[i]->subnode[j]->type != 1) /* Not a current  subnode */
          {
            printf(" *** Error Line %d in file %s\n",
                   strm->line_nbr,strm->filename);
            printf("    *** SUBNODE of wrong type:\n %s\n\n",inline);
            *errflag = 1;
            continue;
          }

        nn[i]->subnode[j]->ref_flag = 1;
      }

    else /* its an error */
      {
        printf(" *** Error Line %d in file %s\n",
               strm->line_nbr,strm->filename);
        printf("    *** SYNTAX ERROR:\n %s\n\n",inline);
```

load_simulation.c

```
            *errflag = 1;
          }
      }
}



set_defaults(simulate)
SIMULATE *simulate;
{
  simulate->dt = .001;
  simulate->tmin = 0.0;
  simulate->tmax = 10.0;
  simulate->time = simulate->tmin;
  simulate->max_iteration = 30;
  simulate->converge = .0000001;
  simulate->delta = .01;
  simulate->delta_min = .001;
  simulate->print_dt = 0.05;
}
```

```
/* make_jacobian.c */
/* Norbert H. Doerry

   15 February 1989

*/
/* This routine creates the jacobian matrix which gives the partial
   of each implicit variable with respect to the individual variables.

   It is created by patching together the individual jacobian submatrices
   of the different elements.

   The hard part about the construction of this matrix is determining
   which column a variable in the submatrix corresponds to.  Here are the
   rules.

   If the variable is attached to a voltage subnode :
     {
        if the variable is attached to the reference subnode,
             ignore it.  (the reference subnode is identically zero)

        else
           find the element in xtab that corresponds to the voltage
           subnode.  That element number is the column.

     }
   else the variable is attached to a current subnode
     {
       if the variable is the first one of a subnode other than
           the reference subnode,
              add the negative of the jacobian element to the
              columns corresponding to the remaining variables

       else
          add the jacobian element to the colum corresponding to the
          element
     }


*/
#include <stdio.h>
#include <math.h>
#include "doerry.h"



/* note : don't need : itab, nitab, n, nnode, simulate */

make_jacob(jacob,xtab,nxtab,itab,nitab,ee,nelm,n,nnode,simulate)
double *jacob;
XTABLE **xtab;
ITABLE **itab;
int nxtab,nitab,nelm,nnode;
ELEMENT **ee;
```

make_jacobian.c

```
NODE **n;
SIMULATE simulate;
{
  int i,j,k,eptr,vptr,vtyp,iptr,rows,col;
  int mult;

  /* zero out the jacobian array */

  for (i = 0 ; i < nxtab * nitab ; i++)
    jacob[i] = 0.0;


  /* step through xtab */

  for (i = 0 ; i < nxtab ; i++)
    {
      for (j = 0  ; j < xtab[i]->nbr ; j++)
        {
          eptr = xtab[i]->e[j];
          vptr = xtab[i]->v[j];
          mult = xtab[i]->mult[j];

          for (k = 0 ; k < ee[eptr]->con.nbr_implicit ; k++)
            {
              jacob[ee[eptr]->con.imp_index[k] + nitab * i] +=
                ee[eptr]->con.jacob_in [k + ee[eptr]->con.nbr_implicit * vptr]
                  * mult;
            }
        }
    }
}
```

penner.h

```
/* penner.h */
/* Norbert H. Doerry

   14  March 1989

   This is an include file which tells the main program where to get
   the proper information for the devices

   *** Modified 11 April 1989 by nhd ****

   added breaker_3p

   *** Modified 15 April 1989 by nhd ****

   added  synch_mach, speed_reg, volt_reg,ind_motor,gas_turbine, source
          integrator

  *** Modified 27 April 1989  by nhd ****

  added volt_meter

*/



typedef int (*FUNCTION_PTR)();

#define NBR_DEV_FILES 2          /* number of device description files */

static char *device_file[] =     /* names of the device description files */
{
  "/mit/13.411/sepsip/three_phase.input",
  "/mit/13.411/sepsip/one_phase.input"
};

static int nbr_device_file[] =
{
  12,                              /* number of devices per file */
  10
};

static char *device_name[] =        /* names of devices */
{
  "t_line_3p",
  "rl_wye",
  "gen_synch_3p",
  "switch_3p",
  "rms",
  "breaker_3p",
  "synch_mach",
  "speed_reg",
  "volt_reg",
  "ind_motor",
  "gas_turbine",
```

```
  "volt_meter",

  "inductor",
  "capacitor",
  "resistor",
  "voltage_source",
  "current_source",
  "diode",
  "switch",
  "pulse_switch",
  "source",
  "integrator"
};

/* device functions for the above device names */

#define F0 t_line_3p
#define F1 rl_wye
#define F2 gen_synch_3p
#define F3 switch_3p
#define F4 rms
#define F4a breaker_3p
#define F4b synch_mach
#define F4c speed_reg
#define F4d volt_reg
#define F4e ind_motor
#define F4f gas_turbine
#define F4g volt_meter

#define F5 inductor
#define F6 capacitor
#define F7 resistor
#define F8 voltage_source
#define F9 current_source
#define F10 diode
#define F11 spst_switch
#define F12 pulse_switch
#define F13 source
#define F14 integrator

int F0 ();
int F1 ();
int F2 ();
int F3 ();
int F4 ();
int F4a();
int F4b();
int F4c();
int F4d();
int F4e();
int F4f();
int F4g();
int F5 ();
int F6 ();
```

penner.h

```
int F7 ();
int F8 ();
int F9 ();
int F10 ();
int F11 ();
int F12 ();
int F13 ();
int F14 ();


static FUNCTION_PTR dev_fnctn[]  =     /* addresses of device functions */
{
  F0,
  F1,
  F2,
  F3,
  F4,
  F4a,
  F4b,
  F4c,
  F4d,
  F4e,
  F4f,
  F4g,
  F5,
  F6,
  F7,
  F8,
  F9,
  F10,
  F11,
  F12,
  F13,
  F14

};
```

print_network.c

```
/* print_network.c */
/* Norbert H. Doerry

   18 January 1989

   This file contains the routine needed to display the network connections
   for the program sepsip.
*/
#include <stdio.h>
#include <math.h>
#include "doerry.h"

print_network(out,nn,nnode)
FILE *out;
NODE **nn;
int nnode;
{
  int i,j,k,cnt;
  char c,inline[MAXCHAR];

  fprintf(out,"\n\n                    NETWORK SUMMARY\n\n");

  cnt = 0;
  for (i = 0 ; i < nnode ; i++)
    {
      fprintf(out,"\n NODE : %s\n",nn[i]->name);
      if (line_counter(&cnt,20,&c,out) == 'q') return;

      for (j = 0 ; j < nn[i]->nbr_subnode ; j++)
        {
          if (nn[i]->subnode[j]->type == 0)
            fprintf(out,"       VOLTAGE ");
          else
            fprintf(out,"       CURRENT ");
          fprintf(out,"SUBNODE : %s\n",nn[i]->subnode[j]->name);
          if (line_counter(&cnt,20,&c,out) == 'q') return;

          for (k = 0 ; k < nn[i]->subnode[j]->nbr_connect ; k++)
            {
              fprintf(out,"           %20s :: %-s\n",
                      nn[i]->subnode[j]->element[k],
                      nn[i]->subnode[j]->variable[k]);
              if (line_counter(&cnt,20,&c,out) == 'q') return;

            }
        }

      if (out != stdout) continue;

      printf(" Enter <RETURN> to continue ... ");
      gets(inline);
      strstrip(inline);
      if (inline[0] == 'q') return;
      if (inline[0] == 'b') i -= 2;   /* go back one node */
```

```
      if (i < -1) i = -1;

    }
}

/* line_counter */
/* this routine keeps track of the number of lines printed on the screen.

   After 'maxcnt' number of lines are listed, the user is prompted to
   hit a return to continue.  'rtnchar' is the first character of the
   line that the user inputs (That is a non white space )
*/

line_counter(cnt,maxcnt,rtnchar,out)
int *cnt,maxcnt;
char *rtnchar;
FILE *out;
{
  char inline[MAXCHAR];

  *rtnchar = NULL;      /* default value */
  (*cnt) += 1;             /* increment counter */

  if (*cnt != maxcnt)
    return 0;

  (*cnt) = 0;

  if (out == stdout) return 0;  /* don't prompt if not printing to screen */

  printf(" Enter <RETURN> to continue ... ");
  gets(inline);
  strstrip(inline);
  *rtnchar = inline[0];
  return (int) inline[0];
}
```

read_device.c

```
/* read_device.c */
/* Norbert H. Doerry

   6 March 1989

*/
/* This routine reads in all the information for a device from an input
   stream.  The following commands are recognized :

      NAME  name of device

      INPUTS [nbr]
        input name 1
        input name 2
        etc

      STATES [nbr]
        state name 1
        state name 2
        etc

      IMPLICIT [nbr]
        implicit variable name 1
        implicit variable name 2
        etc

      EXTERNAL IN [nbr]
        type : extern in name 1
        type : extern in name 2
        etc

      EXTERNAL OUT [nbr]
        type : extern out name 1
        type : extern out name 2
        etc

      PARAMETERS [nbr]
        parameter name 1
        parameter name 2
        etc

      END

This routine returns

      0   successful read of data
      1   encountered bad data, but able to recover
      2   reached EOF before END statement


NOTE: THIS FILE MUST BE LINKED TO

ioliba.c
```

read_device.c

```
*/
#include <stdio.h>
#include "doerry.h"

int read_device(d,in,typ)
DEVICE *d;
FILE *in;
int typ;
{
  char inline[MAXCHAR];
  char command[MAXCHAR];
  char lines[4][MAXCHAR];
  int ans,cnt,name_flag,ncnt;
  float flota[2];
  int i;
  char *calloc(),*malloc();

  ans = 0;
  name_flag = 0;

  /* initialize the d array */

  d->type              = typ;
  d->nbr_inputs        = 0;
  d->nbr_states        = 0;
  d->nbr_implicit      = 0;
  d->nbr_ext_in        = 0;
  d->nbr_ext_out       = 0;
  d->nbr_param         = 0;


  while(1)
    {
      if (fgets(inline,MAXCHAR,in) == NULL)
         return 2;

      parse(inline,(char *) lines, (int) MAXCHAR, (int) 4, &cnt);

      if (cnt == 0) continue;  /* skip blank lines */

      if (lines[0][0] == '\n') continue; /* skip lines beginning with CR */
      if (lines[0][0] == '!' ) continue; /* skip lines beginning with !  */
      if (lines[0][0] == '#' ) continue; /* skip lines beginning with #  */

      if (cnt > 1 && strncmpa(lines[0],"NAME",3) == 0)
        {
          if (name_flag == 1) free(d->name);
          name_flag = 1;
          strsplit(inline,command,1,strlen(inline));
          strstrip(command);
          d->name = (char *) malloc((unsigned) strlen(command) + 1);
          strcpy(d->name,command);
        }
```

```
      else if (cnt > 1 && strncmpa(lines[0],"INPUTS",3) == 0)
        {
          Stofa(lines[1],flota,&ncnt,1);
          if ((int) flota[0] > 0 && ncnt == 1)
            {
               /* see if inputs already allocated */

              if (d->nbr_inputs > 0)
                {
                  for (i = 0 ; i < d->nbr_inputs ; i++)
                    free( (char *) d->input_name[i]);
                  free( (char *) d->input_name);
                }

              /* update number of inputs */

              d->nbr_inputs = (int) flota[0];

              /* allocate the pointer array */

              d->input_name = (char **) calloc( (unsigned) d->nbr_inputs,
                                                sizeof(char *));

              /* read in the input names */

              for (i = 0 ; i < d->nbr_inputs ; i++)
                {
                  if (fgets(command,MAXCHAR,in) == NULL)
                    return 2;

                  strstrip(command);

                  if (command[0] == '!' || command[0] == '#' ||
                      command[0] == NULL)
                    {
                      i--;
                      continue;
                    }

                  d->input_name[i] = (char *)
                    calloc( (unsigned) strlen(command) + 1,
                            sizeof(char));
                  strcpy(d->input_name[i],command);
                }

            }

        }

      else if (cnt > 1 && strncmpa(lines[0],"STATES",3) == 0)
        {
          Stofa(lines[1],flota,&ncnt,1);
          if ((int) flota[0] > 0 && ncnt == 1)
            {
```

```
            /* see if states already allocated */

            if (d->nbr_states > 0)
              {
                for (i = 0 ; i < d->nbr_states ; i++)
                  free( (char *) d->state_name[i]);
                free( (char *) d->state_name);
              }

            /* update number of states */

            d->nbr_states = (int) flota[0];

            /* allocate the pointer array */

            d->state_name = (char **) calloc( (unsigned) d->nbr_states,
                                              sizeof(char *));

            /* read in the state names */

            for (i = 0 ; i < d->nbr_states ; i++)
              {
                if (fgets(command,MAXCHAR,in) == NULL)
                  return 2;
                strstrip(command);


                if (command[0] == '!' || command[0] == '#' ||
                    command[0] == NULL)
                  {
                    i--;
                    continue;
                  }


                d->state_name[i] = calloc( (unsigned) strlen(command) + 1,
                                           sizeof(char));
                strcpy(d->state_name[i],command);
              }

          }

        }

      else if (cnt > 1 && strncmpa(lines[0],"IMPLICIT",3) == 0)
        {
          Stofa(lines[1],flota,&ncnt,1);
          if ((int) flota[0] > 0 && ncnt == 1)
            {
               /* see if implicit already allocated */

              if (d->nbr_implicit > 0)
                {
                  for (i = 0 ; i < d->nbr_implicit ; i++)
```

```
                free( (char *) d->implicit_name[i]);
              free( (char *) d->implicit_name);
            }

          /* update number of implicit */

          d->nbr_implicit = (int) flota[0];

          /* allocate the pointer array */

          d->implicit_name =
            (char **) calloc( (unsigned) d->nbr_implicit,sizeof(char *));

          /* read in the implicit names */

          for (i = 0 ; i < d->nbr_implicit ; i++)
            {
              if (fgets(command,MAXCHAR,in) == NULL)
                return 2;
              strstrip(command);


              if (command[0] == '!' || command[0] == '#' ||
                  command[0] == NULL)
                {
                  i--;
                  continue;
                }

              d->implicit_name[i] =
                calloc( (unsigned) strlen(command) + 1,sizeof(char));
              strcpy(d->implicit_name[i],command);
            }

        }

    }

  else if (cnt > 2 && strncmpa(lines[0],"EXTERNAL",3) == 0
           && strncmpa(lines[1],"INPUT",3) == 0)
    {
      Stofa(lines[2],flota,&ncnt,1);
      if ((int) flota[0] > 0 && ncnt == 1)
        {
          /* see if ext_in already allocated */

          if (d->nbr_ext_in > 0)
            {
              for (i = 0 ; i < d->nbr_ext_in ; i++)
                free( (char *) d->ext_in_name[i]);
              free( (char *) d->ext_in_name);
              free( (char *) d->type_ext_in);
            }
```

```
/* update number of ext_in */

d->nbr_ext_in = (int) flota[0];

/* allocate the pointer array */

d->ext_in_name = (char **) calloc( (unsigned) d->nbr_ext_in,
                                   sizeof(char *));
d->type_ext_in =   (int *) calloc( (unsigned) d->nbr_ext_in,
                                   sizeof(int)   );

/* read in the ext_in names */

for (i = 0 ; i < d->nbr_ext_in ; i++)
  {
    if (fgets(command,MAXCHAR,in) == NULL)
      return 2;

    /* get first non space */

    strstrip(command);


    if (command[0] == '!' || command[0] == '#' ||
        command[0] == NULL)
      {
        i--;
        continue;
      }


    /* get type */

    if (strncmpa(command,"BOOLEAN",3) == 0)
      d->type_ext_in[i] = BOOLEAN;

    else if (strncmpa(command,"SWITCH",3) == 0)
      d->type_ext_in[i] = SWITCH;

    else if (strncmpa(command,"INTEGER",3) == 0)
      d->type_ext_in[i] = INTEGER;

    else if (strncmpa(command,"FLOAT",3) == 0)
      d->type_ext_in[i] = FLOAT;

    else
      {  /* error reading type */
        ans = 1;
        d->type_ext_in[i] = FLOAT;
      }

    /* get name */

    strsplit(command,command,1,MAXCHAR);
```

```
                strstrip(command);

                d->ext_in_name[i] = calloc( (unsigned) strlen(command) + 1,
                                            sizeof(char));
                strcpy(d->ext_in_name[i],command);
              }

          }

      }
    else if (cnt > 2 && strncmpa(lines[0],"EXTERNAL",3) == 0
             && strncmpa(lines[1],"OUTPUT",3) == 0)
      {
        Stofa(lines[2],flota,&ncnt,1);
        if ((int) flota[0] > 0 && ncnt == 1)
          {
             /* see if ext_out already allocated */

            if (d->nbr_ext_out > 0)
              {
                for (i = 0 ; i < d->nbr_ext_out ; i++)
                  free( (char *) d->ext_out_name[i]);
                free( (char *) d->ext_out_name);
                free( (char *) d->type_ext_out);
              }

            /* update number of ext_out */

            d->nbr_ext_out = (int) flota[0];

            /* allocate the pointer array */

            d->ext_out_name =
              (char **) calloc( (unsigned) d->nbr_ext_out,sizeof(char *));
            d->type_ext_out =
              (int *) calloc( (unsigned) d->nbr_ext_out,sizeof(int)   );

            /* read in the ext_out names */

            for (i = 0 ; i < d->nbr_ext_out ; i++)
              {
                if (fgets(command,MAXCHAR,in) == NULL)
                  return 2;

                /* get first non space */

                strstrip(command);



                if (command[0] == '!' || command[0] == '#' ||
                    command[0] == NULL)
                  {
                    i--;
```

```
                continue;
              }


            /* get type */

            if (strncmpa(command,"BOOLEAN",3) == 0)
              d->type_ext_out[i] = BOOLEAN;

            else if (strncmpa(command,"SWITCH",3) == 0)
              d->type_ext_out[i] = SWITCH;

            else if (strncmpa(command,"INTEGER",3) == 0)
              d->type_ext_out[i] = INTEGER;

            else if (strncmpa(command,"FLOAT",3) == 0)
              d->type_ext_out[i] = FLOAT;

            else
              {  /* error reading type */
                ans = 1;
                d->type_ext_out[i] = FLOAT;
              }

            /* get name */

            strsplit(command,command,1,MAXCHAR);
            strstrip(command);

            d->ext_out_name[i] =
              calloc( (unsigned) strlen(command) + 1,sizeof(char));
            strcpy(d->ext_out_name[i],command);
          }

        }

      }

    else if (cnt > 1 && strncmpa(lines[0],"PARAMETERS",3) == 0)
      {
        Stofa(lines[1],flota,&ncnt,1);
        if ((int) flota[0] > 0 && ncnt == 1)
          {
             /* see if param already allocated */

            if (d->nbr_param > 0)
              {
                for (i = 0 ; i < d->nbr_param ; i++)
                  free( (char *) d->param_name[i]);
                free( (char *) d->param_name);
              }

            /* update number of param */
```

```
            d->nbr_param = (int) flota[0];

            /* allocate the pointer array */

            d->param_name = (char **) calloc( (unsigned) d->nbr_param,
                                              sizeof(char *));

            /* read in the param names */

            for (i = 0 ; i < d->nbr_param ; i++)
              {
                if (fgets(command,MAXCHAR,in) == NULL)
                  return 2;
                strstrip(command);


                if (command[0] == '!' || command[0] == '#' ||
                    command[0] == NULL)
                  {
                    i--;
                    continue;
                  }

                d->param_name[i] = calloc( (unsigned) strlen(command) + 1,
                                           sizeof(char));
                strcpy(d->param_name[i],command);
              }

          }

        }

      else if (cnt > 0 && strncmpa(lines[0],"END",3) == 0)
        {
          if (name_flag == 0) ans = 1;
          return ans;
        }

      else
        ans = 1;     /* Encountered bad data */
    }
}
```

```
/* read_element.c */
/* Norbert H. Doerry

   25 October 1988

   This file contains a function for reading an element description
   from a file.

   The form of the device description is:

        DEVICE   ELEMENT
          PARAMETER_NAME  PARAMETER
               | |          | |
              \| |/        \| |/
               \ /          \ /
          END

   where

         DEVICE  is the name of the device type (i.e. resistor)
         ELEMENT is the specific element name (i.e. R1)
         PARAMETER NAME is the name of the DEVICE PARAMETER (i.e. R)
         PARAMETER is the value of the Parameter (i.e. 100 )


     This routine returns:

        0  successful read of data
        1  encountered bad data, but able to recover
        2  encountered EOF before END Statement
       -1  incomplete definition of parameters, defaulted to zero
       -2  unable to find device type


  NOTE:  This file must be linked to

      ioliba.c

*/

#include <stdio.h>
#include "doerry.h"

int read_element(e,strm,serial,dev,ndev)

ELEMENT *e;         /* element structure to hold data */
STREAM_PTR *strm;   /* pointer to stream structure    */
int serial;         /* serial number of element       */
DEVICE **dev;       /* array of device descriptions   */
int ndev;           /* number of device descriptions  */

{
  FILE *in;        /* input stream to read data from */
  char inline[MAXCHAR],line[MAXCHAR];
```

```
  int i,j,k,ncnt,flag,dp,ans;
  float flota[2];
  char *calloc(),*malloc();
  int *param_flag;
  char command[MAXCHAR];

  ans = 0;

  in = strm->in;  /* set input stream */

  flag = 0;

  while (flag == 0)
    {
      strm->line_nbr += 1;   /* increment line counter */

      if (fgets(inline,MAXCHAR,in) == NULL)
        return 2;

      /* strip inline of leading and trailing blanks */

      strstrip(inline);

      /* if comment or blank line, read in another line  */

      if (inline[0] == NULL || inline[0] == '!' || inline[0] == '#')
        continue;

      /* strip off the device name */

      strextract(inline,line,1,MAXCHAR);

      /* find which device we are talking about */

      for (dp = 0 ; dp < ndev && strcmp(dev[dp]->name,line)  != 0 ; dp++);

      /* see if did not find device */

      if (dp == ndev)
        {
          /* if didn't find device, set the element name to the
             input line for further decoding in parent routine */

          e->name = (char *) calloc((unsigned) strlen(inline) + 1,
                                    sizeof(char));
          strcpy(e->name,inline);
          return -2;
        }

      /* done with loop */

      flag = 1;
    }
```

```
  /* get name of element */

  strextract(inline,line,2,MAXCHAR);

  /* save element name */

  e->name = (char *) calloc((unsigned) strlen(line) + 1,sizeof(char));
  strcpy(e->name,line);

  /* set Device pointer */

  e->device = dev[dp];

  /* set the serial number */

  e->serial = serial;

  /* set connection type pointer */

  e->con.type_ext_in  = dev[dp]->type_ext_in;
  e->con.type_ext_out = dev[dp]->type_ext_out;

  /* set number of elements and allocate arrays */

  e->con.nbr_inputs   = dev[dp]->nbr_inputs;
  e->con.nbr_states   = dev[dp]->nbr_states;
  e->con.nbr_implicit = dev[dp]->nbr_implicit;
  e->con.nbr_ext_in   = dev[dp]->nbr_ext_in;
  e->con.nbr_ext_out  = dev[dp]->nbr_ext_out;
  e->con.nbr_param    = dev[dp]->nbr_param;

  if (dev[dp]->nbr_inputs > 0)
    {
      e->con.in  = (double *) calloc((unsigned) dev[dp]->nbr_inputs   ,
                                     sizeof(double));

      e->con.init_in  = (double *) calloc((unsigned) dev[dp]->nbr_inputs   ,
                                     sizeof(double));

      if (dev[dp]->nbr_implicit > 0)
        e->con.jacob_in = (double *) calloc((unsigned) dev[dp]->nbr_inputs *
                                            dev[dp]->nbr_implicit ,
                                            sizeof(double));

    }

  if (dev[dp]->nbr_states > 0)
    {
      e->con.state      = (double *) calloc((unsigned) dev[dp]->nbr_states,
                                            sizeof(double));
      e->con.old_state  = (double *) calloc((unsigned) dev[dp]->nbr_states,
                                            sizeof(double));
      e->con.init_state = (double *) calloc((unsigned) dev[dp]->nbr_states,
                                            sizeof(double));
```

```
    }
  if (dev[dp]->nbr_implicit > 0)
    {
      e->con.implicit = (double *) calloc((unsigned) dev[dp]->nbr_implicit,
                                          sizeof(double));
      e->con.imp_index = (int    *) calloc((unsigned) dev[dp]->nbr_implicit,
                                          sizeof(int));
    }
  if (dev[dp]->nbr_ext_in   > 0)
    {
      e->con.ext_in      = (double *) calloc((unsigned) dev[dp]->nbr_ext_in,
                                           sizeof(double));
      e->con.init_ext_in = (double *) calloc((unsigned) dev[dp]->nbr_ext_in,
                                           sizeof(double));
    }

  if (dev[dp]->nbr_ext_out  > 0)
    {
      e->con.ext_out  = (double *) calloc((unsigned) dev[dp]->nbr_ext_out,
                                          sizeof(double));
    }

  if (dev[dp]->nbr_param    > 0)
    e->con.param    = (double *) calloc((unsigned) dev[dp]->nbr_param,
                                        sizeof(double));

  /* read in parameters */

  /* allocate parameter flag array (used to ensure all the parameters
     are specified) */

  if (dev[dp]->nbr_param   > 0)
    param_flag    = (int *) calloc((unsigned) dev[dp]->nbr_param,sizeof(int));

    /* ensure param_flag is initialized to zero along with parameters */

  for (i = 0 ; i < dev[dp]->nbr_param ; i++ )
    e->con.param[i] = param_flag[i] = 0;

  flag = 0;

  while (flag == 0)
    {

      strm->line_nbr += 1;

      if (fgets(inline,MAXCHAR,in) == NULL)
        return 2;

      /* strip inline of leading and trailing blanks */

      strstrip(inline);

      /* if comment or blank line, read in another line  */
```

```
      if (inline[0] == NULL || inline[0] == '!' || inline[0] == '#')
        continue;

      /* stop if end statement */

      slctouc(inline,command);
      strstrip(command);
      if (strcmp(command,"END") == 0)    /* NOTE : case insensitive */
        {
          flag = 1;
          continue;
        }

      /* check param_name[] to find out which parameter should be read in */

      strextract(inline,command,1,MAXCHAR);

      for (i = 0 ; i < dev[dp]->nbr_param &&
           strcmp(command,dev[dp]->param_name[i])  != 0 ; i++);

      /* if did not find parameter, set ans = 1 (invalid data read in) */

      if (i == dev[dp]->nbr_param)
        {
          ans = 1;
          continue;
        }

      /* get parameter value */

      strextract(inline,command,2,MAXCHAR);

      Stofa(command , flota , &ncnt , 1);

      if (ncnt == 1)
        {
          e->con.param[i] = flota[0];
          param_flag[i] = 1;
        }
      else
        ans = 1;      /* invalid data read in */

    }

  /* check all the parameter flags */

  for (i = 0 ; i < dev[dp]->nbr_param ; i++)
    if (param_flag[i] == 0)
      {
        /* found a parameter that wasn't initialized */

        ans = -1;  /* supplied default value */
      }
```

```
  return ans;

}
```

```
/* read_network.c */
/* Norbert H. Doerry

   6 December 1988

   This file contains the code for reading the network connections
   between the various elements.  The form for the netork description
   is

   NODE NODE_NAME
      [r]t[n]:SUB_NAME = [val] = ELM1:NAME = ELM2:NAME = ELM3:NAME     etc.
            ||                ||
           \||/              \||/
            \/                \/
      END

 -------
  NODE_NAME is the name of the node

  [r]   is an optional character 'r' to make the subnode a reference
        subnode.
  t is either a 'v' for voltage law node or an 'i' for a current law node
  [n]   is the number of variables to be equated beginning with the specified
      one.  (usually 3 for three phase)  If omitted, assumed equal to 1.
  SUB_NAME is the name of the subnode

  [val] For a voltage subnode, the program tries to convert the first
        entry into a number.  If successful, the value becomes the initial
        value if not a reference node, and the actual value if the node
        is a reference node.

  ELM1 is the first element name being connected
  NAME is the name of the variable being connected

 -------

 Element names can not begin with a numeral or a punctuation sign.  Elements
 should not have colons or equal signs in their name.

 ****  Modified 17 April 1989 *****

    Fixed the nbr_c bug.                -nhd


*/

#include <stdio.h>
#include <math.h>
#include "doerry.h"

#define DEBUG 0


read_network(strm,node,e,nelm,errflag)
```

```
STREAM_PTR *strm;
NODE *node;
ELEMENT **e;
int nelm;
int *errflag;   /* set to one if detect a fatal error */
{
  char inline[MAXCHAR],line[MAXCHAR],*sline;
  FILE *in;
  int i,ans,flag;
  char *calloc();
  SUBNODE **s_node,*e_node;
  NODE *new_node;

  in = strm->in;

  /* initialize s_node */

  s_node = (SUBNODE **) calloc(1,sizeof(SUBNODE *));
  *s_node = (SUBNODE *) calloc(1,sizeof(SUBNODE));
  (*s_node)->nbr_connect = 0;
  (*s_node)->last = NULL;  /* indicator that this is the first one */

  /* get first line */

  ans = 0;   /* flag for all loaded normally */

  while (1)
    {
      strm->line_nbr += 1;

      if (fgets(inline,MAXCHAR,in) == NULL)
        return 2;

      /* strip inline of leading and trailing blanks, etc */

      strstrip(inline);

      /* if comment or blank line, read in next line */

      if (inline[0] == NULL || inline[0] == '!' || inline[0] == '#')
        continue;

      /* see if node */

      if (strncmpa(inline,"NODE",3) != 0)
        {
          /* if not, pass the offending line back to the main routine */

          node->name = calloc(strlen(inline) + 1 , sizeof(char));
          strcpy(node->name,inline);
          return -1;   /* not a NODE command */

        }
```

```
      /* Have a node,  lets get its name */

      strsplit(inline,line,1,MAXCHAR);
      strstrip(line);
      for (i = 0 ; line[i] != ' ' && line[i] != '\t' && line[i] != NULL ; i++);
      line[i] = NULL;

      node->name = calloc(strlen(line) + 1, sizeof(char));

      strcpy(node->name,line);

      break;
    }

  /* get the node information */

  while (1)
    {
      strm->line_nbr += 1;

      if (fgets_multiple(&sline,MAXCHAR,in) == NULL)
        return 2;

      /* strip sline of leading and trailing blanks, etc */

      strstrip(sline);

      /* if comment or blank line, read in next line */

      if (sline[0] == NULL || sline[0] == '!' || sline[0] == '#')
        continue;

      /* see if end statement */

      if (strcmpa(sline,"END") == 0)
        break;

      /* read the sub nodes in */

      if (flag = read_sub_node(strm,sline,s_node,e,nelm,1,errflag) != 0)
        {
          ans = 1;
          *errflag = 1;
          if (flag == -3 || flag == -4) return flag;  /* out of memory error */
          continue;
        }

      /* free sline */

      free(sline);

    }
```

```
  /* count the number of subnodes */

  e_node = *s_node;

  for (i = 0 ; e_node->last != NULL ; i++ , e_node = e_node->last);

  node->nbr_subnode = i;

  /* allocate array */

  node->subnode = (SUBNODE **) calloc( i ,sizeof (SUBNODE *));

  if (node->subnode == NULL)
    return -3;

  /* fill the array */

  for ( ; i > 0 ; i--)
    {
      node->subnode[i-1] = (*s_node)->last;
      *s_node = (*s_node)->last;
    }
  return ans;
}
read_sub_node(strm,inlines,s_node,e,nelm,pe,errflag)
char *inlines;
STREAM_PTR *strm;
SUBNODE **s_node;
ELEMENT **e;
int nelm;
int *errflag;  /* if equal to one, indicates fatal error */
int pe;     /* print error flag  : 1 yes   0 no */
{
  char *inline;
  int nbr_snode,i,j,k,l,kk,nbr_c,ncnt;
  char *make_str();
  char line[MAXCHAR],name[MAXCHAR];
  float flota[2];
  SUBNODE *t_node;  /* tempory sub node */
  char *make_str();
  double val;

  inline = make_str(inlines);  /* copy inlines into inline */

  nbr_c = count_char(inline,'=');  /* count equal signs */

  if (DEBUG) printf("%s || nbr_c = %d\n",inlines,nbr_c);


  if (nbr_c < 1)   /* didn't get an equal sign */
    {
      if (pe)
        {
```

```
          printf(" *** Error Line %d in file %s\n",
                 strm->line_nbr,strm->filename);

          printf("    *** '=' not found in node description ***\n %s\n",
               inlines);
        }
      *errflag = 1;
      return 1;
    }

  (*s_node)->nbr_connect = nbr_c;

  /* find type of subnode and number of subnodes */

  strstrip(inline);

  /* see if the subnode is a reference subnode */

  if (inline[0] == 'r' || inline[0] == 'R')
    {
      inline[0] = ' ';
      (*s_node)->ref_flag = 1;
      strstrip(inline);
    }
  else
    (*s_node)->ref_flag = 0;

  /* find out if a voltage or current subnode */

  if (inline[0] == 'v' || inline[0] == 'V')
    {
      inline[0] = ' ';  /* set to space */
      (*s_node)->type = 0;
    }
  else if (inline[0] == 'i' || inline[0] == 'I')
    {
      inline[0] = ' ';  /* set to space */
      (*s_node)->type = 1;
    }
  else
    {
      if (pe)
        {
          printf(" *** Error Line %d in file %s\n",
                 strm->line_nbr,strm->filename);

          printf("    *** Improper SUBNODE TYPE in node description ***\n");
          printf(" %s\n",inlines);
        }
      *errflag = 1;
      return 1;
    }

  /* find number of subnodes */
```

```
  Stofa(inline,flota,&ncnt,1);

  nbr_snode = (ncnt == 0) ? 1 : flota[0];

  /* allocate arrays */

  (*s_node)->name     = (char  *) calloc(MAXCHAR,sizeof (char));
  (*s_node)->element  = (char **) calloc(nbr_c,sizeof (char *));
  (*s_node)->variable = (char **) calloc(nbr_c,sizeof (char *));
  (*s_node)->elm_ptr  = (int   *) calloc(nbr_c,sizeof (int   ));
  (*s_node)->var_ptr  = (int   *) calloc(nbr_c,sizeof (int   ));

  if ((*s_node)->element == NULL || (*s_node)->variable == NULL ||
      (*s_node)->elm_ptr == NULL || (*s_node)->var_ptr  == NULL ||
      (*s_node)->name     == NULL)
    {
      if (pe)
        {

          printf(" *** Error Line %d in file %s\n",
                 strm->line_nbr,strm->filename);
          printf("    *** Out of MEMORY\n");
        }
      *errflag = 1;
      return -3;
    }

  /* get rid of number */

  for (i = 0 ; inline[i] != NULL && inline[i] != ':' ; i++)
    inline[i] = ' ';

  /* ensure got to a colon */

  if (inline[i] != ':')
    {
      if (pe)
        {
          printf(" *** Error Line %d in file %s\n",
                 strm->line_nbr,strm->filename);

          printf("    ***  in node description ***\n");
          printf(" %s\n",inlines);
        }
      *errflag = 1;
      return 1;
    }

  inline[i] = ' ';
  strstrip(inline); /* remove leading blanks */

  /* read in name of subnode */
```

```
  for (i = 0 ; inline[i] != NULL && inline[i] != '=' ; i++)
    {
      line[i] = inline[i];
      inline[i] = ' ';
    }
  line[i] = NULL;
  if (inline[i] == '=') inline[i] = ' ';
  strstrip(line);

  /* ignore everything after first space or tab and before = */

  for (i = 0 ; line[i] != ' ' && line[i] != '\t' && line[i] != NULL ; i++)
    (*s_node)->name[i] = line[i];
  (*s_node)->name[i] = NULL;

  strstrip(inline);

  /* start reading in the elements */

  for (i = 0 , j = 0 ; i < nbr_c ; i++ , j++)
    {

      /* get the element name */

      for (k = 0 ; inline[j] != '=' && inline[j] != NULL && inline[j] != ':';
           j++,k++)
        {
          line[k] = inline[j];
        }
      line[k] = NULL;
      strstrip(line);

      /* see if it is possibly an ititial value */

      if (i == 0 && (*s_node)->type == 0)
        {
          Stoda(line,&val,&ncnt,1);
          if (ncnt == 1) /* successful conversion */
            {
              (*s_node)->init_volt = val;
              i--;
              nbr_c--;  /* decrement number of connections by one */

              (*s_node)->nbr_connect = nbr_c; /* do it for real */

              continue;
            }
        }


      /* allocate the string */

      (*s_node)->element[i] = (char *) calloc(strlen(line) + 1,sizeof(char));
```

```
      strcpy((*s_node)->element[i],line);

      /* get the variable name */

      if (inline[j] != ':')
        {
          *errflag = 1;
          if (pe)
            {

              printf(" *** Error Line %d in file %s\n",
                     strm->line_nbr,strm->filename);

              printf("    *** Missing variable name for element %s ***\n %s\n",
                   (*s_node)->element[i],inlines);
            }
          (*s_node)->variable[i] = NULL;
          continue;
        }

      j++;   /* increment pointer to character in inline past colon */


      /* copy next element/variable into line */

      for (k = 0 ; inline[j] != '=' && inline[j] != NULL; j++,k++)
        {
          line[k] = inline[j];
        }
      line[k] = NULL;
      strstrip(line);

      /* allocate the string */

      (*s_node)->variable[i] = (char *) calloc(strlen(line) + 1,sizeof(char));

      /* see if out of memory */

      if ((*s_node)->variable[i] == NULL)
        {
          *errflag = 1;
          if (pe)
            {

              printf(" *** Error Line %d in file %s\n",
                     strm->line_nbr,strm->filename);
              printf("    *** Out of MEMORY ***\n");
            }
          return -3;
        }

      /* copy the string */

      strcpy((*s_node)->variable[i],line);
```

```
/* find the elements in the arrays and pass pointers etc.  */


/* find out which element we belong to */

for (l = 0 ; l < nelm && strcmp(e[l]->name,(*s_node)->element[i]) != 0 ;
     l++)
  if (DEBUG) printf("%s || %s\n",e[l]->name,(*s_node)->element[i]);

/* see if the element name wasn't found */

if (l == nelm)
  {
    *errflag = 1;
    if (pe)
      {
        printf(" *** Error Line %d in file %s\n",
               strm->line_nbr,strm->filename);
        printf("    *** Can not recognize ELEMENT %s ***\n",
               (*s_node)->element[i]);
      }
    continue;
  }

/* assign pointer */

(*s_node)->elm_ptr[i] = l;

/* turn flag on for element */

e[l]->flag = 1;

/* find the variable name */


/* look for input variables */

for (k = 0 ; l < nelm && k < e[l]->device->nbr_inputs &&
     strcmp((*s_node)->variable[i],e[l]->device->input_name[k]) != 0;
     k++)
  if (DEBUG) printf("*** %s || %s\n",(*s_node)->variable[i],
                    e[l]->device->input_name[k]);

if ( l < nelm && k <  e[l]->device->nbr_inputs)
  {
    (*s_node)->var_ptr[i] = k;
    continue;
  }


if (pe)
  {
```

```
        printf(" *** Error Line %d in file %s\n",
               strm->line_nbr,strm->filename);

        printf("    *** Variable %s not recognized for element %s ***\n",
                  (*s_node)->variable[i],(*s_node)->element[i]);
      }

    *errflag = 1;
    (*s_node)->var_ptr[i] = 0;

  }

/* equate follow on variables with new sub_nodes */

strcpy(name,(*s_node)->name);

for (i = 1 ; i < nbr_snode ; i++)
  {

    /* allocate new arrays */

    t_node = (SUBNODE *) calloc(1,sizeof(SUBNODE));
    if (t_node == NULL) return -3;

    t_node->last = *s_node;

    t_node->nbr_connect = (*s_node)->nbr_connect;

    /* allocate arrays */

    t_node->name    =  (char *) calloc(MAXCHAR,sizeof(char));

    t_node->elm_ptr  = (int  *) calloc(t_node->nbr_connect,
                                       sizeof(int));
    t_node->var_ptr  = (int  *) calloc(t_node->nbr_connect,
                                       sizeof(int));
    t_node->element = (char **) calloc(t_node->nbr_connect,
                                       sizeof(char *));
    t_node->variable= (char **) calloc(t_node->nbr_connect,
                                       sizeof(char *));

    /* see if out of memory problems */

    if (t_node->elm_ptr  == NULL || t_node->var_ptr == NULL ||
        t_node->element == NULL ||
        t_node->variable == NULL || t_node->name    == NULL) return -3;

    /* set reference flag to zero */

    t_node->ref_flag = 0;

    /* copy data */
```

```
      t_node->type = (*s_node)->type;
      strcpy(line,name);

      /* append  an underscore  followed by a 'b' or 'c' etc  to the subnode
         name */

      sprintf(t_node->name,"%s_%c",line,'a' + i);

      for (l = 0 ; l < t_node->nbr_connect ; l++)
        {
          t_node->element[l]  = make_str((*s_node)->element[l]);
          t_node->elm_ptr[l]  = (*s_node)->elm_ptr[l];

          t_node->var_ptr[l] = (*s_node)->var_ptr[l] + 1;

          /* see if too many points */

          if (t_node->var_ptr[l] >=
              e[t_node->elm_ptr[l]]->device->nbr_inputs)
            return -4;

          /* copy the variable name */

          k = t_node->var_ptr[l];
          kk = t_node->elm_ptr[l];

          t_node->variable[l] = make_str((e[kk])->device->input_name[k]);


        }
      /* update s_node */

      *s_node = t_node;

    }


  /* allocate new arrays */

  t_node = (SUBNODE *) calloc(1,sizeof(SUBNODE));
  if (t_node == NULL) return -3;

  t_node->last = *s_node;
  *s_node = t_node;

  /* free inline */

  free(inline);

  return 0;
}
```

```
/* count_char returns the number of times a character occurs in a string */
count_char(s,c)
char *s,c;
{
  int i,j;

  for (i = j = 0 ; s[i] != NULL ; i++)
    if (s[i] == c) j++;
  return j;
}

/* make_str  allocates a string array and copies the argument passed to it  */
char *make_str(s)
char *s;
{
  char *calloc();
  char *ans;

  ans = calloc(strlen(s) + 1 , sizeof(char));

  if (ans == NULL) return NULL;  /* out of memory error */

  strcpy(ans,s);

  return ans;
}


typedef struct String
{
  char *s;
  struct String *next;
}
STRING_;

fgets_multiple(outline,max,in)
char **outline;
int max;
FILE *in;
{
  char *out;
  STRING_ strt,*ptr;
  char *inline;
  int flag,len;

  ptr = &strt;
  ptr->next = NULL;

  inline = calloc((unsigned) max + 1 , sizeof(char));

  flag = 1;

  while(flag)
    {
```

```
      /* if read to EOF, send back EOF */

      if (fgets(inline,max,in) == NULL)
        {
          free(inline);
          return NULL;
        }


      strstrip(inline);

      /* see if its a comment */

      if (inline[0] == '!' || inline[0] == '#' || inline[0 ] == NULL)
        continue;

      /* see if a continuation marker is present */

      flag = 0;

      if (inline[strlen(inline) - 1] == '\\')
        {
          inline[strlen(inline) - 1] = NULL;
          flag = 1;
        }

      if (strcmp(inline + strlen(inline) - 3 , "...") == 0)
        {
          inline[strlen(inline) - 3] = NULL;
          flag = 1;
        }

      /* store the string and allocate a new array */

      ptr->s = make_str(inline);
      ptr->next = (STRING_ *) calloc((unsigned) 1 , sizeof (STRING_));
      ptr = ptr->next;
      ptr->next = NULL;

    }

  for (ptr = &strt,len = 1 ; ptr->next != NULL ; ptr = ptr->next)
    len += strlen(ptr->s) + 1;

  *outline = (char *) calloc((unsigned)len, sizeof (char));

  (*outline)[0] = NULL;

  for (ptr = &strt ; ptr->next != NULL ; ptr = ptr->next)
    {
      strcat(*outline," ");
      strcat(*outline,ptr->s);
    }
```

```
  free(inline);
  return (*outline)[0];
}
```

```
/* sepsip.c */
/* 11 November 1988

       ****  version 1.0  27 March 1989  ****

   Norbert H. Doerry

   Shiboard Electrical Plant SImulation Program


   **** modified 9 April 1989 ****

   Fixed bug that caused utility line of menu to be printed twice

   **** modified 10 April 1989 ****

   Added ability to enter stdout as a filename for writing files
   from the get_filename function

*/


#include <stdio.h>
#include "doerry.h"
#include "penner.h"
#define CLEARSCREEN system("clear")
#define VERSION 1.0
#define VERSION_DATE "27 March 1989"
#define DIR "ls -al"
#define CMD "/mit/nhdoerry/diffeq/thesis/sepsip_util"

#define DEBUG 0

#define WRITE_FILE 1
#define READ_FILE 0


main(argc,argv)
int argc;
char **argv;
{
  FILE *in;
  extern char *device_file[];
  extern int  nbr_device_file[];
  extern FUNCTION_PTR dev_fnctn[];
  extern char *device_name[];
  STREAM_PTR *strm;
  DEVICE **dev;
  ELEMENT **e;
  QUEUE **queue;
  char *calloc();
  char inline[MAXCHAR],infile[MAXCHAR];
  int i,k,typ,ndev,nelm,nqueue;
  NODE **nn;
```

```
  int nnode,errflag;
  SIMULATE simulate;
  XTABLE **xtab;
  ITABLE **itab;
  int nxtab,nitab;
  PRINT_VAR pv;
  int sim_flag;

  /* print Header */

  printf("\n\n WELCOME TO SEPSIP\n\n");
  printf(" Version %5.2f   :  Version Date %s\n\n",VERSION,VERSION_DATE);

  /* initialize errflag to 0 (no errors) */

  errflag = 0;

  /* allocate the storage for the device descriptions */

  for (i = ndev = 0 ; i < NBR_DEV_FILES ; i++)
    ndev += nbr_device_file[i];

  dev = (DEVICE **) calloc((unsigned) ndev,sizeof(DEVICE *));

  for (i = 0 ; i < ndev ; i++)
    dev[i] = (DEVICE *) calloc(1,sizeof(DEVICE));


  /* read in the device descriptions */

  for (i = typ = 0 ; i < NBR_DEV_FILES ; i++)
    {
      if ( (in = fopen(device_file[i],"r")) == NULL)
      {
        errflag = 1;
        printf(" *** Unable to Open Device File : %s\n",device_file[i]);
        continue;
      }

      k = load_device(dev + typ , nbr_device_file[i] , in , typ ,
                  dev_fnctn , device_name , ndev);

      typ += nbr_device_file[i];
    }

  /* exit program if any of the device files are not present */

  if (errflag)
    {
      printf(" *** Program Terminated\n\n\n");
      exit();
    }

  /* see if there is a filename specified in argv */
```

```
  if (argc > 1)
    {
      strcpy(infile,argv[1]);
      strcpy(inline,"l ");
      strcat(inline,argv[1]);
    }
  else
    infile[0] = NULL;


  if (infile[0] != NULL)
    errflag = load_file(infile,&e,&nelm,dev,ndev,inline,&nn,&nnode,
                        &queue,&nqueue,&simulate,&xtab,&nxtab,
                        &itab,&nitab,&pv);
  else
    errflag = 1;

  /* should have all the devices and elements read in */

  sim_flag = 0;  /* simulation has not occurred */

  while(1)
    {

      printf("\n\n SEPSIP Commands : \n");
      if (errflag == 0 && sim_flag == 1)
        {
          printf("   c  Continue Simulation\n");
        }
      printf("   d  Display Data\n");
      if (errflag == 0)
        {
          printf("   e  Edit Simulation Parameters\n");
        }
      printf("   f  File Options\n");
      printf("   l  Load New Input File\n");
      printf("   q  Quit\n");
      if (errflag == 0)
        {
          printf("   s  Conduct Simulation\n");
        }
      printf("   u  Utilities\n");
      printf(" Enter Command : ");

      gets(inline);
      strstrip(inline);

      if (inline[0] == 'c' && errflag == 0 && sim_flag == 1)
        run_simulation(inline,e,nelm,nn,nnode,queue,nqueue,xtab,nxtab,
                       itab,nitab,&pv,&simulate,1);

      else if (inline[0] == 'd')
        display_data(inline,dev,ndev,e,nelm,nn,nnode,queue,
```

```
                    nqueue,errflag,stdout);

      else if (inline[0] == 'e' && errflag == 0)
        edit_simulate(e,nelm,nn,nnode,&simulate,&pv,inline);

      else if (inline[0] == 'f')
        file_options(inline,e,nelm,nn,nnode,&queue,&nqueue,&simulate,
                    &pv,xtab,nxtab,itab,nitab,errflag);

      else if (inline[0] == 'l')
        {
          errflag = load_file(infile,&e,&nelm,dev,ndev,inline,&nn,&nnode,
                              &queue,&nqueue,&simulate,&xtab,&nxtab,
                              &itab,&nitab,&pv);
          sim_flag = 0;  /* reset the simulation flag */
        }

      else if (inline[0] == 'q' || inline[0] == 'Q')
        exit();

      else if (inline[0] == 's' && errflag == 0)
        {
          run_simulation(inline,e,nelm,nn,nnode,queue,nqueue,xtab,nxtab,
                         itab,nitab,&pv,&simulate,0);
          sim_flag = 1; /* set the simulation flag */
        }

      else if (inline[0] == 'u')
        utilities(CMD,inline);

    }
}

int load_file(infile,ee,nelm,dev,ndev,inline,nn,nnode,q,nq,simulate,
              xtab,nxtab,itab,nitab,pv)
char *infile;
ELEMENT ***ee;
int *nelm;
DEVICE **dev;
int ndev;
char *inline;
NODE ***nn;
int *nnode;
QUEUE ***q;
int *nq;
SIMULATE *simulate;
XTABLE ***xtab;
int *nxtab;
ITABLE ***itab;
int *nitab;
PRINT_VAR *pv;

{
  STREAM_PTR *strm;
```

```
  FILE *in;
  char filename[MAXCHAR];
  int i,errflag,flag;

  flag = 0;
  strsplit(inline,filename,1,MAXCHAR);  /* grab the filename if specified */

  errflag = 1; /* ensure loop occurs at least once */
  while (errflag != 0)
    {
      printf("\n\n");
      errflag = 0;  /* reinitialize errflag */

      if (filename[0] != NULL)
        {
          in = fopen(filename,"r");

          i = (in == NULL) ?
            get_filename(infile,&in,READ_FILE,"SEPSIP INPUT",0) : 0;
        }
      else
        i = get_filename(infile,&in,READ_FILE,"SEPSIP INPUT",0);


      if (i != 0)
        {
          return flag;   /* one if an error , zero otherwise */
        }

      filename[0] = NULL;  /* ensure that second time through, a file name
                              is prompted for */

      /* initialize starting structures */

      strm = (STREAM_PTR *) calloc(1,sizeof(STREAM_PTR));
      strm->in  = in;
      strm->last = NULL;  /* indicator that this is the first stream */
      strcpy(strm->filename,infile);
      strm->line_nbr = 0;


      /* set defaults */

      set_defaults(simulate);

      pv->next = NULL;

      i = load_element(&strm,ee,nelm,dev,ndev,&errflag);
      if (i != 0)  /* hit EOF before all data was read in */
        {
          flag = errflag = 1;
          fclose(strm->in);
          continue;
        }
```

```
      else
        flag = (errflag != 0) ? 1 : 0 ;

      i = load_network(&strm,nn,nnode,*ee,*nelm,&errflag);
      if (i != 0)
        {
          flag = errflag = 1;
          fclose(strm->in);
          continue;
        }
      else
        flag = (errflag != 0) ? 1 : 0 ;

      strcpy(inline,(*nn)[*nnode]->name);

      /* see if elements and nodes multiply defined */

      check_name(*nn,*nnode,*ee,*nelm,&errflag);

      i = load_initial(&strm,*ee,*nelm,*nn,*nnode,inline,&errflag,0);
      if (i != 0)
        {
          flag = errflag = 1;
          fclose(strm->in);
          continue;
        }
      else
        flag = (errflag != 0) ? 1 : 0 ;

      load_simulation(&strm,*ee,*nelm,*nn,*nnode,q,nq,simulate,pv,&errflag);
      flag = (errflag != 0) ? 1 : 0 ;

      if (DEBUG)
        {
          for (i = 0 ; i < *nq ; i++)
            printf("elm = %d , var = %d , val = %f , time = %f\n",(*q)[i]->elm,
                   (*q)[i]->var,(*q)[i]->value,(*q)[i]->time);
        }

      setup_simulation(*nn,*nnode,*ee,*nelm,*simulate,xtab,nxtab,
                       itab,nitab,&errflag);
      flag = (errflag != 0) ? 1 : 0 ;

      if (DEBUG)
        {
          printf("***********\n");
          for (i = 0 ; i < *nq ; i++)
            printf("elm = %d , var = %d , val = %f , time = %f\n",(*q)[i]->elm,
                   (*q)[i]->var,(*q)[i]->value,(*q)[i]->time);
        }
    }

  return flag;
```

```
}

get_filename(filename,stream,type,string,flag)
char *filename;       /* default name of file */
FILE **stream;         /* io stream */
int type;             /* = 0 for read;  = 1 for write */
char *string;         /* string to prompt user with */
int flag;             /* try to load file immediately if non zero */
{
  char inline[MAXCHAR],direction[2],command[15];

  direction[0] = (type == WRITE_FILE) ? 'w' : 'r';
  direction[1] = NULL;

  if (type == READ_FILE) strcpy(command,"Read From");
  else strcpy(command,"Write To");

  strstrip(filename);

  while(flag == 0 || filename[0] == NULL)
    {
      if (filename[0] == NULL)
      printf(" Enter %s file name : ",string);
      else
      printf(" Enter %s file name (Default %s) : ",string,filename);

      gets(inline);
      strstrip(inline);

      if (inline[0] == 'q' && inline[1] == NULL)
      return -1;

      if (inline[0] == NULL)
      break;

      if (inline[0] == '?')
      system(DIR);
      else
      {
        strcpy(filename,inline);
        break;
      }
    }

  /* see if still NULL filename */

  if (filename[0] == NULL)
    {
      if (type)
      {
        *stream = stdout;
        return 0;
      }
      else return -1;
```

```
    }

  /* see if write file and filename is stdout */

  if (strcmp(filename,"stdout") == 0 && type)
    {
      *stream = stdout;
      return 0;
    }

  /* try to open the file */

  while ((*stream = fopen(filename,direction)) == NULL)
    {
      while(1)
      {
        printf(" Cannot %s %s : Enter %s file name : ",
               command,filename,string);
        gets(inline);
        strstrip(inline);

        if (inline[0] == 'q' && inline[1] == NULL)
          return -1;

        if (inline[0] == NULL)
          break;

        if (inline[0] == '?')
          system(DIR);
        else
          {
            strcpy(filename,inline);
            break;
          }
      }
    }

  return 0;
}


display_data(inline,dev,ndev,e,nelm,nn,nnode,q,nq,errflag,out)
char *inline;
DEVICE **dev;
int ndev;
ELEMENT **e;
int nelm;
NODE **nn;
int nnode;
QUEUE **q;
int nq;
int errflag;
FILE *out;
{
```

```
  char line[MAXCHAR],cmd;
  int flag;

  strcpy(line,inline);
  line[0] = ' ';  /* strip off first character */
  strstrip(line);
  cmd  = line[0];

  if (cmd != 'd' && cmd != 'D' && cmd != 'e' && cmd != 'E' && cmd != 'n')
    flag = 1;
  else
    flag = 0;

  while (1)
    {
      if (flag)
        {
          printf("\n DISPLAY DATA\n");

          printf("   d  Display Device Summary\n");
          printf("   D  Display Device Data\n");
          if (errflag == 0)
            {
              printf("   e  Display Element Summary\n");
              printf("   E  Display Element Data\n");
              printf("   n  Display Network Summary\n");
            }
          printf("   q  Quit\n");
          printf("   w  Write Device Data File\n");

          printf(" Enter Command : ");
          gets(line);
          strstrip(line);
          cmd = line[0];

        }


      if (cmd == 'd')
        device_summary(dev,ndev,stdout);

      else if (cmd == 'D')
        display_device(dev,ndev,line);

      else if (cmd == 'e' && errflag == 0)
        element_summary(e,nelm,out,line);

      else if (cmd == 'E' && errflag == 0)
        display_element(e,nelm,line,q,nq);

      else if (cmd == 'n' && errflag == 0)
        print_network(out,nn,nnode);

      else if (cmd == 'q')
```

```
        return;

      else if (cmd == 'w')
        dump_device(dev,ndev,line);
      else flag = 1;

      if (flag == 0) return;
    }


}

/* use utility menu driver to execute program */

/* cmd is the the command name of the utility menu driving program */
/* inline is the string input from the user */

utilities(cmd,inline)
char *inline,*cmd;
{
  char *command,*calloc();

  command = calloc(strlen(inline) + strlen(cmd) + 10,sizeof(char));

  strcpy(command,cmd);

  inline[0] = ' ';
  strstrip(inline);
  strcat(command," ");
  strcat(command,inline);

  system(command);
}
```

```
/* setup_simulation.c */
/* Norbert H. Doerry

   28 February 1989

   This routine sets up the state variable list for conducting the simulation.

   The variable list contains the following variables:

         | |         |    Node Voltages                          |
         |x|          =  | Variables attached to Current Nodes.**  |
         | |         |                                           |


         ** All the variables except the first one assigned to the
         current node.

         These variables are related by a set of implicit equations of
         the form:

         _
         F (|x|) = |I| --> 0

         where |I| is a vector of 'implicit variables' That should be
         driven to zero.

         Obviously, the order of |x| should be the same as the order of
         |I|.

         At a Voltage Node, the voltage variables attached to it are all
         set equal to the Node Voltage.  The Node Voltage is the variable
         that is allowed to vary.

         At a Current Node, The first variable attached to it is set equal
         to the negative of the sum of the remaining variables.  If there
         are no more remaining variables, then the first variable is set
         equal to zero.  The remaining variables attached to the current
         node are allowed to vary.  If the Current subnode is the reference
         subnode, then the first variable is kept as a separate variable.


         Here are the definitions of the XTABLE and ITABLE structures :

           typedef struct Xtable
           {
             int nbr;              || number of variables tied to this variable
             int *e;               || array of element indexes
             int *v;               || array of variable indexes
             int *mult;            || array of multipliers for variables
           }
           XTABLE;


           typedef struct Itable
           {
```

```
            int e;                  || index to element array
            int i;                  || index to implicit variable array
          }
          ITABLE;



*/


#include <stdio.h>
#include <math.h>
#include "doerry.h"
#define DEBUG 0

setup_simulation(nn,nnode,ee,nelm,simulate,xtab,nxtab,itab,nitab,errflag)
NODE **nn;
int nnode;
ELEMENT **ee;
int nelm;
SIMULATE simulate;
XTABLE ***xtab;
int *nxtab;
ITABLE ***itab;
int *nitab;
int *errflag;
{
  int i,j,k,l;
  int nbr_x,nbr_i;

  /* count the variables */

  nbr_x = nbr_i = 0;

  for (i = 0 ; i < nnode ; i++)
    {
      for (j = 0 ; j < nn[i]->nbr_subnode ; j++)
        {

          /* skip reference voltage subnode */

          if (nn[i]->subnode[j]->ref_flag == 1 &&
              nn[i]->subnode[j]->type == 0)
            continue;

          /* add extra variable for current reference subnode */

          if (nn[i]->subnode[j]->ref_flag == 1 &&
              nn[i]->subnode[j]->type == 1)
            nbr_x++;

          /* add up the variables : one             for voltage subnode
                                    nbr_connect - 1 for current subnode */
```

```
        if (nn[i]->subnode[j]->type == 0)  /* voltage subnode */
          nbr_x += 1;
        else /* current subnode */
          nbr_x += nn[i]->subnode[j]->nbr_connect - 1;
      }
  }


/* count the implicit  variables */

for (i = 0 ; i < nelm ; i++)
  {
    /* skip element if it is not used */

    if (ee[i]->flag == 0) continue;

    nbr_i += ee[i]->con.nbr_implicit;
  }

if (DEBUG)
  printf("nbr_i = %d :: nbr_x = %d\n",nbr_i,nbr_x);


/* ensure number of implict variables and node variables are the same */

if (nbr_i != nbr_x)
  {
    printf(" *** SYSTEM DEFINITION ERROR :\n");
    printf
      ("    *** Unequal number of variables and implicit variables ***\n");
    printf("    *** nbr_x = %d || nbr_i = %d\n",nbr_x,nbr_i);
    *errflag = 1;
    return;
  }

/* allocate arrays */

*xtab = (XTABLE **) calloc((unsigned) nbr_i, sizeof(XTABLE *));
*itab = (ITABLE **) calloc((unsigned) nbr_i, sizeof(ITABLE *));
for (i = 0 ; i < nbr_i ; i++)
  {
    (*xtab)[i] = (XTABLE *) calloc((unsigned) 1, sizeof(XTABLE));
    (*itab)[i] = (ITABLE *) calloc((unsigned) 1, sizeof(ITABLE));
  }

/* fill the arrays */

k = 0;

for (i = 0 ; i < nnode ; i++)
  {
    for (j = 0 ; j < nn[i]->nbr_subnode ; j++)
      {
```

```
        /* skip reference voltage subnode */

        if (nn[i]->subnode[j]->ref_flag == 1 &&
            nn[i]->subnode[j]->type == 0)
          continue;

        /* see if reference current subnode */

        if (nn[i]->subnode[j]->ref_flag == 1 &&
            nn[i]->subnode[j]->type == 1)
          {
            for (l = 0 ; l < nn[i]->subnode[j]->nbr_connect ; l++)
              {
                if (DEBUG) printf(" k = %d\n",k);

                (*xtab)[k]->nbr = 1;
                (*xtab)[k]->e   = (int *) calloc((unsigned) 1,sizeof(int));
                (*xtab)[k]->v   = (int *) calloc((unsigned) 1,sizeof(int));
                (*xtab)[k]->mult= (int *) calloc((unsigned) 1,sizeof(int));

                (*xtab)[k]->e[0]   = nn[i]->subnode[j]->elm_ptr[l];
                (*xtab)[k]->v[0]   = nn[i]->subnode[j]->var_ptr[l];
                (*xtab)[k]->mult[0]= 1;
                k++;
              }
            continue;
          }

        /* see if a normal voltage subnode */
        /* ***there is some redundancy here, could save memory by
           using the pointers instead of copying */

        if (nn[i]->subnode[j]->type == 0)  /* voltage subnode */
          {
            if (DEBUG) printf(" k = %d\n",k);

            (*xtab)[k]->nbr = nn[i]->subnode[j]->nbr_connect;
            (*xtab)[k]->e   = (int *) calloc((unsigned) (*xtab)[k]->nbr,
                                             sizeof(int));
            (*xtab)[k]->v   = (int *) calloc((unsigned) (*xtab)[k]->nbr,
                                             sizeof(int));
            (*xtab)[k]->mult= (int *) calloc((unsigned) (*xtab)[k]->nbr,
                                             sizeof(int));

            for (l = 0 ; l < (*xtab)[k]->nbr ; l++)
              {
                (*xtab)[k]->e[l]   = nn[i]->subnode[j]->elm_ptr[l];
                (*xtab)[k]->v[l]   = nn[i]->subnode[j]->var_ptr[l];
                (*xtab)[k]->mult[l]= 1;
              }
            k++;
          }
        else /* current subnode */
          {
```

```
            for (l = 1 ; l < nn[i]->subnode[j]->nbr_connect ; l++)
              {
                if (DEBUG) printf(" k = %d\n",k);


                (*xtab)[k]->nbr = 2;
                (*xtab)[k]->e   = (int *) calloc((unsigned) 2,sizeof(int));
                (*xtab)[k]->v   = (int *) calloc((unsigned) 2,sizeof(int));
                (*xtab)[k]->mult= (int *) calloc((unsigned) 2,sizeof(int));

                (*xtab)[k]->e[0]   = nn[i]->subnode[j]->elm_ptr[l];
                (*xtab)[k]->v[0]   = nn[i]->subnode[j]->var_ptr[l];
                (*xtab)[k]->mult[0]= 1;

                (*xtab)[k]->e[1]   = nn[i]->subnode[j]->elm_ptr[0];
                (*xtab)[k]->v[1]   = nn[i]->subnode[j]->var_ptr[0];
                (*xtab)[k]->mult[1]= -1;

                k++;
              }
          }
      }
  }


/* save the implicit variable pointers */

for (i = 0,k = 0 ; i < nelm ; i++)
  {
    /* skip element if it is not used */

    if (ee[i]->flag == 0) continue;

    for (j = 0 ; j < ee[i]->con.nbr_implicit ; j++)
      {
        if (DEBUG) printf(" k = %d\n",k);

        (*itab)[k]->e = i;
        (*itab)[k]->i = j;
        ee[i]->con.imp_index[j] = k;
        k++;
      }
  }

*nitab = nbr_i;
*nxtab = nbr_x;

}
```

```
/* simulate.c */
/* Norbert H. Doerry

   14 March 1989

   This routine performs the actual simulation of the electrical system

   The procedure is :

   A.  Initialize
         old state variables to their initial values
         set current variable initial guesses to the value specified.
         set voltage variable initial guesses to the value specified for
               the first voltage specified at a voltage subnode.

   B.  Balance system

         B1.  Calculate the implicit variables, see if convergence is
                    satisfied. (if satisfied, go to C)

         B2.  Manufacture Jacobian.
                 B2a.  From function calls
                 B2b.  By varying inputs to system and allowing to change

         B3.  Solve system of equations with Jacobian and Implicit variables

         B4.  Apply correction to voltage and current guesses.

   C.  Print output variables

   D.  Increment time counter (if after TMAX, then end)

   E.  See if external input variables have changed, if so change them

   F.  Set old state variables to present state variables

   G.  Go to B

   NOTE:  eventually may have to include tests for switches which will]
     effectively  change the system.  For example,  for a diode, if the
     the voltage drop is less than .6 volts or the current is going int
     the wrong direction, then one of the implicit equations will drive
     one of the currents to zero.  If the voltage drop is greater than
     .6 volts and the current is in the right direction, then the implicit
     equation is set equal to .6 - the voltage drop.

     After a switch is thrown, the system will have to be rebalanced.  If
     more than one switch can be thrown at a time is unknown.


     *** 27 March 1989 ***

     The above statements are probably not true,  By properly rotating
     the axes of the coordinate systems, I think one can properly define
```

```
     the implicit variables such that you will get convergence. (As long
     as the characteristic is continuous) -nhd


     *** Revision a :  29 March 1989 ***

     Added fflush(out) statements so that the output file will have
     something in it if the program crashes.

*/
#include <stdio.h>
#include <math.h>
#include "doerry.h"

#define BIG 1.0e+16
#define DEBUG 0

run_simulation(line,ee,nelm,n,nnode,qq,nq,xtab,nxtab,itab,nitab,pv,
               simulate,flag)
char *line;  /* this is the command line */
ELEMENT **ee;
int nelm;
NODE **n;
int nnode;
QUEUE **qq;
int nq;
XTABLE **xtab;
int nxtab;
ITABLE **itab;
int nitab;
SIMULATE *simulate;
PRINT_VAR *pv;
int flag;  /* flag = 0, start at beginning, flag = 1 continue to new TMAX */
{
  FILE *out;
  char filename[MAXCHAR];
  int i,j,k;
  double *jacob,*implicit,*var;
  char *calloc();
  double implicit_error(),square_error,start;
  double temp;

  if (DEBUG)
    {
      for (i = 0 ; i < nq ; i++)
        printf("elm = %d , var = %d , val = %f , time = %f\n",qq[i]->elm,
               qq[i]->var,qq[i]->value,qq[i]->time);
    }

  jacob    = (double *) calloc(nxtab * nxtab , sizeof(double));
  implicit = (double *) calloc(nxtab         , sizeof(double));
  var      = (double *) calloc(nxtab         , sizeof(double));

  if (jacob == NULL) return 1;  /* out of memory  error */
```

```
  /* see if want to write to a file */

  strsplit(line,filename,1,MAXCHAR);

  if (filename[0] != 0)
    {
      out = fopen(filename,"w");
      if (out == NULL) out = stdout;
    }
  else
    out = stdout;

  /* initialize the simulation to initial values if flag == 0 */

  if (flag == 0)
    {
      initialize_simulation(ee,nelm,n,nnode,*simulate);
      simulate->time = simulate->tmin;
    }

  /* perform the simulation */

  for (k = 0, start = simulate->time  ; simulate->time <= simulate->tmax ;
       simulate->time = start + (double)(++k) * simulate->dt)
    {

      /* check the external input variable queue */

      check_queue(ee,nelm,qq,nq,simulate->time);

      for (i = 0 ; i <= simulate->max_iteration ; i++)
        {
          /* calculate the implicit variables */

          /* first time, balance the system */

/*          if (k == 0)
            calc_implicit(ee,nelm,0.0);
          else
*/            calc_implicit(ee,nelm,simulate->dt);

          /* calculate jacobian submatrices if calc_implicit didn't
             already do so */

          for (j = 0 ; j < nelm ; j++)
            {
              if (ee[j]->con.jacob_switch == 1)
                continue;

/*              if (k == 0)
                {
                  temp = simulate->dt;
                  simulate->dt = 0.0;
```

```
                elm_jacob(ee[j],*simulate);
                simulate->dt = temp;
              }
            else
*/              elm_jacob(ee[j],*simulate);
          }

        /* find the mean square error */

        square_error = implicit_error(ee,nelm);

        /* see if have met convergence test */

        if (square_error < simulate->converge)
          break;

        /* make the jacobian matrix */

        make_jacob(jacob,xtab,nxtab,itab,nitab,ee,nelm,n,nnode,*simulate);

        /* make the implicit variable vector */

        make_implicit(ee,nelm,itab,nitab,implicit);

        /* solve the system of equations */

        if (gauss_eliminate(nxtab,jacob,implicit,var) != 0)
          {
            printf(" *** SINGULAR SYSTEM MATRIX at time %f\n",
                   simulate->time);
            return 0;
          }

        /* note that the contents of jacob and implicit were
           destroyed by gauss_eliminate */


        /* apply the corrections to the variables */

        update_variables(ee,nelm,n,nnode,xtab,nxtab,var);


      }

    /* see if failed the convergence test */

    if (i >= simulate->max_iteration)
      {
        printf(" *** CONVERGENCE TEST FAILED at time %f\n",simulate->time);
        return 0;
      }

    /* print the output variables if at the proper time */
```

```
      print_output(out,ee,nelm,n,nnode,pv,simulate);

      /* set the old_state variables equal to the state variable */

      for (i = 0 ; i < nelm ; i++)
        {
          for (j = 0 ; j < ee[i]->con.nbr_states ; j++)
            ee[i]->con.old_state[j] = ee[i]->con.state[j];
        }

    }

  /* free the jacobian array along with the other two vectors */

  free(jacob);
  free(implicit);
  free(var);

  if (out != stdout) fclose(out);

}

initialize_simulation(ee,nelm,n,nnode,simulate)
ELEMENT **ee;
int nelm;
NODE **n;
int nnode;
SIMULATE simulate;
{
  int i,j,k,eptr,vptr,typ;
  double node_volt,current,sum;

  /* initialize state variables and external input variables */

  for (i = 0 ; i < nelm ; i++)
    {
      for (j = 0 ; j < ee[i]->con.nbr_states ; j++)
        ee[i]->con.old_state[j] = ee[i]->con.init_state[j];

      for (j = 0 ; j < ee[i]->con.nbr_ext_in ; j++)
        ee[i]->con.ext_in[j] = ee[i]->con.init_ext_in[j];


      /* initialize jacobian matrices to zero */

      for (j = 0 ; j < ee[i]->con.nbr_implicit ; j++)
        for (k = 0 ; k < ee[i]->con.nbr_inputs ; k++)
          ee[i]->con.jacob_in[j + ee[i]->con.nbr_implicit * k] = 0.0;


    }

  /* initialize the input variables */
```

```
  for (i = 0 ; i < nnode ; i++)
    {
      for (j = 0 ; j < n[i]->nbr_subnode ; j++)
        {
          if (n[i]->subnode[j]->type == 0) /* voltage law */
            {
              eptr = n[i]->subnode[j]->elm_ptr[0];
              vptr = n[i]->subnode[j]->var_ptr[0];
              node_volt = n[i]->subnode[j]->init_volt;

              ee[eptr]->con.in[vptr] = node_volt;

              for (k = 1 ; k < n[i]->subnode[j]->nbr_connect ; k++)
                {
                  eptr = n[i]->subnode[j]->elm_ptr[k];
                  vptr = n[i]->subnode[j]->var_ptr[k];
                  ee[eptr]->con.in[vptr] = node_volt;
                }
            }
          else  /* current law */
            {
              for (k = 1 ,sum = 0; k < n[i]->subnode[j]->nbr_connect ; k++)
                {
                  eptr = n[i]->subnode[j]->elm_ptr[k];
                  vptr = n[i]->subnode[j]->var_ptr[k];
                  current  = ee[eptr]->con.init_in[vptr];

                  ee[eptr]->con.in[vptr] = current;
                  sum += current;
                }

              /* get pointer of first variable */

              eptr = n[i]->subnode[j]->elm_ptr[0];
              vptr = n[i]->subnode[j]->var_ptr[0];

              /* initialize the current for the first variable */
              /* If the first variable is the reference current node/subnode,
                 then it is an independent state variable,  otherwise, set
                 it equal to the negative sum of the other currents entering
                 the subnode  */

              ee[eptr]->con.in[vptr] = /* see if reference node */
                (n[i]->subnode[j]->ref_flag == 1) ?
                  ee[eptr]->con.init_in[vptr] : -sum;



            }
        }
    }
}
```

```
calc_implicit(ee,nelm,dt)
ELEMENT **ee;
int nelm;
double dt;
{
  int i;

  int (*f)();


  for (i = 0 ; i < nelm ; i++)
    {
      if (ee[i]->flag == 0)
        continue;

      f = ee[i]->device->f;

      (*f)(ee[i],dt);
    }
}

/* implicit_error returns the mean square value for the implicit variable */

double implicit_error(ee,nelm)
ELEMENT **ee;
int nelm;
{
  int i,j,k,nbr;
  double err;

  nbr = 0;
  err = 0.0;

  for (i = 0 ; i < nelm ; i++)
    {
      if (ee[i]->flag == 0)
        continue;

      for (k = 0 ; k < ee[i]->con.nbr_implicit ; k++)
        {
          if (fabs(ee[i]->con.implicit[k]) < BIG)
            err += ee[i]->con.implicit[k] * ee[i]->con.implicit[k];
          else
            err += BIG;
          nbr++;
        }
    }

  if (nbr != 0)
    return  (err / (double) nbr);
  else
    return 0.0;
```

```
}

make_implicit(ee,nelm,itab,nitab,implicit)
ELEMENT **ee;
int nelm;
ITABLE **itab;
int nitab;
double *implicit;
{
  int i,eptr,iptr;

  for (i = 0 ; i < nitab ; i++)
    {
      eptr = itab[i]->e;
      iptr = itab[i]->i;

      implicit[i] = ee[eptr]->con.implicit[iptr];
    }
}


/* check_queue updates the external inputs.  It assumes that
   the queue array is in time order */

check_queue(ee,nelm,qq,nq,time)
ELEMENT **ee;
int nelm;
QUEUE **qq;
int nq;
double time;
{
  int i;

  for (i = 0 ; i < nq && qq[i]->time <= time ; i++)
    {
      ee[qq[i]->elm]->con.ext_in[qq[i]->var] = qq[i]->value;

      if (DEBUG) printf("elm = %d , var = %d , val = %f , time = %f\n",
                        qq[i]->elm,qq[i]->var,qq[i]->value,qq[i]->time);

    }
}


update_variables(ee,nelm,n,nnode,xtab,nxtab,var)
ELEMENT **ee;
int nelm;
NODE **n;
int nnode;
XTABLE **xtab;
int nxtab;
double *var;
{
  int i,j;
```

```
  /* update all the variables */

  for (i = 0 ; i < nxtab ; i++)
    {
      for (j = 0 ; j <  xtab[i]->nbr ; j++)
        {
            ee[xtab[i]->e[j]]->con.in[xtab[i]->v[j]] -=
              var[i] * (double) xtab[i]->mult[j];
        }
    }



}
static double last_display;

print_output(out,ee,nelm,nn,nnode,pv,simulate)
ELEMENT **ee;
int nelm;
NODE **nn;
int nnode;
PRINT_VAR *pv;
SIMULATE *simulate;
FILE *out;
{
  int i,j,k;
  extern double last_display;
  PRINT_VAR *temp;

  /* print out header */

  if (simulate->time == simulate->tmin)
    {
      fprintf(out,"    time    ");

      for (temp = pv->next ; temp != NULL ; temp = temp->next)
        {
          if (temp->typ != 2) /* external output or input*/
            fprintf(out,"%-12s ",ee[temp->e]->name);
          else  /* print node name */
            fprintf(out,"%-12s ",nn[temp->e]->name);
        }
      fprintf(out,"\n");

      fprintf(out,"            ");

      for (temp = pv->next ; temp != NULL ; temp = temp->next)
        {
          if (temp->typ == 0) /* external output */
            fprintf(out,"%-12s ",ee[temp->e]->device->ext_out_name[temp->v]);
          else if (temp->typ == 1) /* external input */
            fprintf(out,"%-12s ",ee[temp->e]->device->ext_in_name[temp->v]);
          else  /* print node subname */
```

```
            fprintf(out,"%-12s ",nn[temp->e]->subnode[temp->v]->name);
        }
      fprintf(out,"\n");

      /* ensure first line gets printed */

      last_display = simulate->time - 2 * simulate->print_dt;

      /* print out variable names  on screen if not printing data
         to the screen */

      if (out != stdout)
        {
          for (temp = pv->next ; temp != NULL ; temp = temp->next)
            {
              if (temp->typ == 0) /* external output*/
                printf("   %12s : %-12s\n",ee[temp->e]->name,
                       ee[temp->e]->device->ext_out_name[temp->v]);
              else if (temp->typ == 1) /* external input */
                printf("   %12s : %-12s\n",ee[temp->e]->name,
                       ee[temp->e]->device->ext_in_name[temp->v]);
              else  /* print node name */
                printf("   %12s : %-12s\n",nn[temp->e]->name,
                       nn[temp->e]->subnode[temp->v]->name);
            }
          printf("\n");
        }

    }


  if (
      (simulate->time - last_display >=
       simulate->print_dt - simulate->dt / 10.0)
      || simulate->time == simulate->tmin
      || simulate->time >= simulate->tmax)
    {
      last_display = simulate->time;
      fprintf(out," %-12.5g ",simulate->time);

      for (temp = pv->next ; temp != NULL ; temp = temp->next)
        {
          if (temp->typ == 0) /* external output */
            fprintf(out,"%-12.5g ",ee[temp->e]->con.ext_out[temp->v]);
          else if (temp->typ == 1) /* external input */
            fprintf(out,"%-12.5g ",ee[temp->e]->con.ext_in[temp->v]);
          else
            /* subnode voltage */
            {
              i = nn[temp->e]->subnode[temp->v]->elm_ptr[0];
              j = nn[temp->e]->subnode[temp->v]->var_ptr[0];
              fprintf(out,"%-12.5g ",ee[i]->con.in[j]);
            }
        }
```

```
      fprintf(out,"\n");

      if (out != stdout)
        {
          printf(".");
          fflush(stdout);
          fflush(out);   /* added as revision a */
        }

    }
}

print_matrix(out,mat,m,n)
FILE *out;
double *mat;
int m;
int n;
{
  int i,j;  /* i is row, j is column */

  fprintf(out,"\n");

  for (i = 0 ; i < m ; i++)
    {
      for (j = 0 ; j < n ; j++)
        fprintf(out," %7.4g",mat[i + j*m]);
      fprintf(out,"\n");
    }
}
```

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943

Thesis
D6326 Doerry
c.1 Computer simulation of shipboard electrical distribution systems.

3 APR 97 36908

Thesis
D6326 Doerry
c.1 Computer simulation of shipboard electrical distribution systems.

Comptuer simulation of shipboard electri
3 2768 000 82195 3
DUDLEY KNOX LIBRARY